1990

A Functional Approach to Model Management.

Lijen Ko
Louisiana State University and Agricultural & Mechanical College

INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

Order Number 9104145

A functional approach to model management

Ko, Lijen, Ph.D.

The Louisiana State University and Agricultural and Mechanical Col., 1990

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

A FUNCTIONAL APPROACH TO MODEL MANAGEMENT

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

Interdepartmental Program in Business Administration

by
Lijen Ko
B.S., Chung Hsing University, 1979
M.S., Louisiana State University, 1986
May 1990

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to my committee members: Drs. Sunkhamay Kundu, Michael H. Peters, James M. Pruett, and Kwei Tang for their guidance and valuable suggestions. I owe a special thank you to Dr. Ye-Sho Chen, the chairman of my committee, for his assistance and advice. I am also grateful to Dr. Daniel B. Rinks for his concern and encouragement.

Finally, I am very thankful to my husband, Pao-Chuan Lin, and my mother for their understanding, love, and support throughout my graduate studies.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

ABSTRACT

A model management system (MMS) is a computer-based system which facilitates the management of a wide range of decision support models (DSMs). It is a vital component of a decision support system which helps decision makers using data and DSMs in an integrated fashion.

It is desirable that an MMS is knowledge-based, flexible, independent and reflecting users' view of a DSM. In addition, an MMS must manage DSMs at both the macro and micro levels. At the macro level, DSMs are described using descriptive attributes to supply users with general problem-solving capabilities. At the micro level, the components and mathematical structure of a DSM are expressed to facilitate DSM formulation.

There are three two-level MMS approaches proposed in the literature. All three approaches suggest describing the details of a DSM using the frame system. However, there are problems using the frame system at the micro level. First, the frame system does not provide facilities for handling data of large volume. Second, there is neither design methodology nor evaluation criteria regarding the design of the frame system. Third, pre-defined frame constructs may not be able to encompass

all DSMs used in a dynamic environment. Finally, designers of the frame system usually lack of the professional skills to design an appropriate structure.

The functional MMS is proposed to overcome the deficiency. It is intended to provide the two-level model management capability with all the desirable features. The macro-level functional MMS is based on first-order logic which is the best-developed knowledge representation methodology so far. At the micro level, the foundation is on relational theory which has proven its usefulness in data management. Additionally, the definitional system used to describe the details of a DSM provides a natural way of developing a DSM in a hierarchical manner.

CHAPTER 1

INTRODUCTION

Presumably business decisions can be improved with timely access to reliable data. However, timely access to good data does not guarantee the success of business. To use data effectively and efficiently most of today's decision makers also need access to decision support models (DSMs). Data and DSMs are two very important assets for decision makers to deal with the complexity and uncertainty of a business problem in a dynamic environment. DSMs, as well as data, have been identified as valuable resources in an organization and must be managed and controlled to improve the operating effectiveness and efficiency.

A DSM is an abstract description of reality which provides a concise framework for analyzing a decision problem in a systematic manner (Geoffrion 1987). In a DSM, some down-to-earth connections in the real world are expressed as mathematical relationship among a set of numeric data. Thus, a DSM is defined as the mathematical associations among a set of numeric elements; a complete DSM description should include both elements and mathematical structure of a DSM.

Due to successful research on solving a lot of well-specified DSMs, it is realized that comprehension has replaced solution as the main obstacle to the use of DSMs (Greenberg 1983; Williams 1985). This trend has been further accentuated by the recent movement of the model management system (MMS). An MMS is a computer-based system which facilitates representing, organizing and manipulating a wide range of DSMs to be used to solve different decision problems in different situations (Elam, Henderson, and Miller 1980; Blanning 1982; Konsynski 1983). In fact, an effective MMS is a vital component of a larger computer-based system, a decision support system (DSS).

A DSS helps a manager use both data and DSMs to make business decisions in an integrated fashion. It is widely accepted that a DSS consists of three major components: a data management system, an MMS and a dialog management system (Sprague 1980). Within the general architecture of a DSS (Figure 1), the MMS must be closely integrated with the data management system because the latter maintains the data referenced in a DSM which is under the control of the former. To allow the close integration between an MMS and a data management system, it is essential that the description of a DSM stored in a model base include all the elements of the DSM.

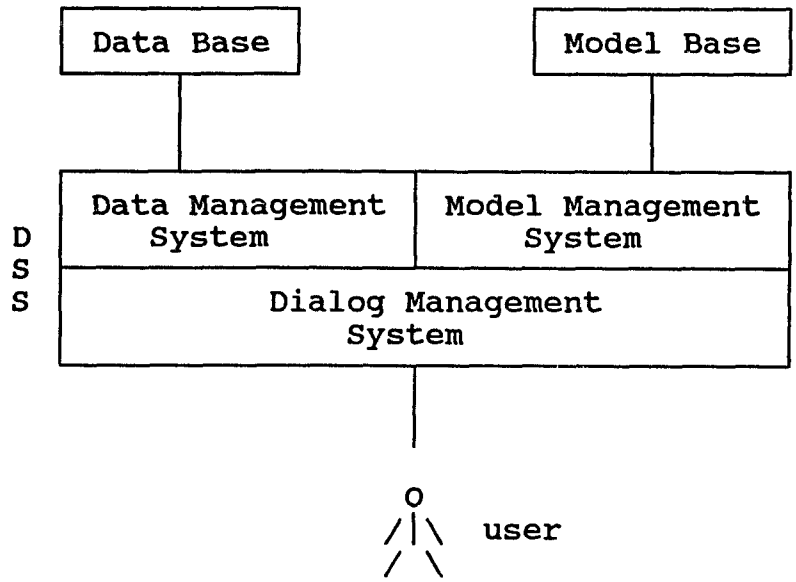


Figure 1. The Architecture of a DSS

Desirable Features of an MMS

An MMS requires several features in order to bridge the gap between users and computers. Several features of an MMS have been proposed in the literature of MMSs (Dolk and Konsynski 1984; Bhargava, Bieber, and Kimbrough 1988; Liang and Jones 1988), and are summarized in Table 1.

Table 1.

Desirable Features of MMSs
1. Knowledge-based,
2. Flexible,
3. Independent, and
4. Reflecting a user's view of a DSM.

The first desirable feature of an MMS is being knowledge-based (Dolk and Konsynski 1984) because the use of knowledge is essential in building a DSM. Using a

knowledge base instead of a model specialist, an MMS brings users, or even decision makers themselves, closer to the computer form of a decision problem and often results in greater user acceptance and satisfaction. In other words, users are more willing to implement the result suggested by the solution of a DSM which is explored, developed and validated by themselves with the help of an intelligent and interactive MMS (Elam and Konsynski 1987).

The second desirable feature of MMSs is flexibility. The MMSs should be sufficiently flexible to encompass a wide range of DSMs (Geoffrion 1987), and to support use of DSMs of different application domains. Furthermore, an MMS must allow DSMs to be combined with the data from various sources: a user, a data base, and/or another DSM.

The third desirable feature of MMSs is independence. An MMS must preserve representational independence not only between the mathematical structure and detailed data of a DSM, but also between DSMs and the solution procedures (Geoffrion 1987). Additionally, the techniques used within an MMS should be independent of application domains; otherwise it becomes impossible to integrate DSMs across different application domains.

Finally, the fourth desirable feature of an MMS is to reflect a user's view of a DSM. As described earlier, a DSM helps decision makers in analyzing a decision

problem. An incomprehensible DSM is of little use to a decision maker because the solution of such a DSM is not understandable and thus is not acceptable (Little 1970). Besides, only when DSMs are comprehensible can the effectiveness of DSMs be evaluated. Therefore, analogous to a data base management system, an MMS must accommodate the user's view of a DSM (Chen 1988) and represent DSMs as cognitive chunks of knowledge which is meaningful to the users (Liang and Jones 1988).

Managing DSMs at the Macro or Micro Level

With the aforementioned features, an MMS can manage DSMs at two different levels: the macro and micro levels (Fedorowicz and Williams 1986). At the macro level, DSMs are viewed as single entities, and are described using some descriptive attributes to supply users with general problem solving capabilities. At the micro level, the components and mathematical structure of a DSM are clearly expressed to facilitate formulating DSMs specifically requested by users.

The purpose of the macro-level model management is to support the decision-making process in selecting useful DSMs to help solving a decision problem. At the macro level, DSMs are often expressed in a meta-language. For example, a DSM may be represented as a list of parameters and decision variables. A transportation DSM may be represented as a list of demands, supplies, unit

transportation costs, and shipment quantities. As such, knowledge about DSMs available to users is restricted to the descriptive attributes incorporated in the meta-language. Consequently, DSMs are not described completely in an MMS so that they are opaque to users.

The purpose of the micro-level model management, on the other hand, is to assist users constructing ad hoc DSMs to solve a decision problem by providing an easy-to-use and easy-to-understand modeling language. The emphasis of model management at the micro level is on expressing elements of a DSM, and the interrelated mathematical relationship (such as equations, inequalities, and so on). Briefly speaking, the micro-level model management is meaningful due to the importance of integrating an MMS with the data management system within a DSS. As mentioned, the close integration of an MMS and data management system requires that DSM descriptions include numeric elements of DSMs. By including complete descriptions of DSMs, a micro-level MMS permits repeated use of standard DSMs, reduces complexity of modeling efforts and improves modeling efficiency.

Statement of the Problem

The choice of a macro-level or micro-level MMS depends on the objective of an MMS. A macro-level MMS is to support selecting a useful DSM; a micro-level MMS, to support constructing a DSM. However, a macro-level or

micro-level MMS is not good enough; a two-level MMS is necessary due to the symbolic and numeric nature of modeling knowledge (Dhar and Croker 1988).

A DSM is, by definition, the mathematical relationship among a set of numeric elements. Nevertheless, a DSM is not purely numeric; it must possess symbolic knowledge about application and limitations. Because of the numeric and symbolic nature of a DSM, an MMS must integrate the knowledge of both types, and reason about the application. Therefore, an MMS is actually an application of coupled systems which link numeric computing and symbolic reasoning models by embedding knowledge of numeric models within the systems (Kitzmiller and Kowalik 1986).

Basically, the macro-level model management employs knowledge of symbolic type. A macro-level MMS perceives DSMs as single entities and guides application of DSMs by representing the symbolic knowledge of DSMs in the form of heuristic rules to solve a decision problem. In contrast, the micro-level model management uses knowledge of numeric type. A micro-level MMS explicitly expresses and extensively utilizes mathematical functions, constraints and other similar numeric descriptors of DSMs when solving a decision problem.

In the literature of MMSs, many approaches have been proposed to deal with the MMS problem. Among them, there

are three approaches (Dolk and Konsynski 1984; Fedorowicz and Williams 1986; Applegate, Konsynski, and Nunamaker 1986) addressing the MMS issue at both the macro and micro levels. At the macro level, these three approaches guide the selection of DSMs by applying various knowledge representation techniques to represent and utilize knowledge of DSMs. At the micro level, the approaches unanimously suggest the frame system for describing the components and mathematical structure of a DSM.

Though having some appealing characteristics (Dolk and Konsynski 1984), the frame system is not a panacea with regard to the MMS problem, especially at the micro level. There are some difficulties of using the frame system to provide the complete descriptions of DSMs. First of all, pre-defined frame constructs may not be able to encompass all kinds of DSMs needed in a dynamic environment. Secondly, DSM builders, who supposedly undertake the task of designing a frame construct, usually lack the professional skills in the design of an appropriate frame structure for representing DSMs. Thirdly, in the field of Artificial Intelligent (AI), there is no existing methodology which is applicable to the design of the frame system, let alone the criteria to evaluate how well a frame system is designed. Finally, the frame system does not provide facilities for handling large volume of data.

Research Methodology

To manage DSMs at both the macro and micro levels, it is proposed that MMS be designed by adopting the functional approach. The functional MMS approach attempts to furnish a two-level conceptual framework for dealing with DSMs, to couple numeric and symbolic knowledge of DSMs, and to preserve the desirable features of MMSs. To overcome the deficiencies of the frame system it also aims to provide a model definition language for users to build and modify their own DSMs in different context based on relational data base theory. Furthermore, the functional MMS applies a uniform design approach for both DSMs and data.

The functional MMS approach can be employed in a generalized DSS architecture (Minch and Burns 1983 - see Figure 2a) which enhances the one shown in Figure 1 with the work by other researchers (Bonczek, Holsapple, and Whinston 1981a). The generalized DSS architecture uses a problem processing system to coordinate and control the operations of the data management system, the MMS and the dialog management system as a whole.

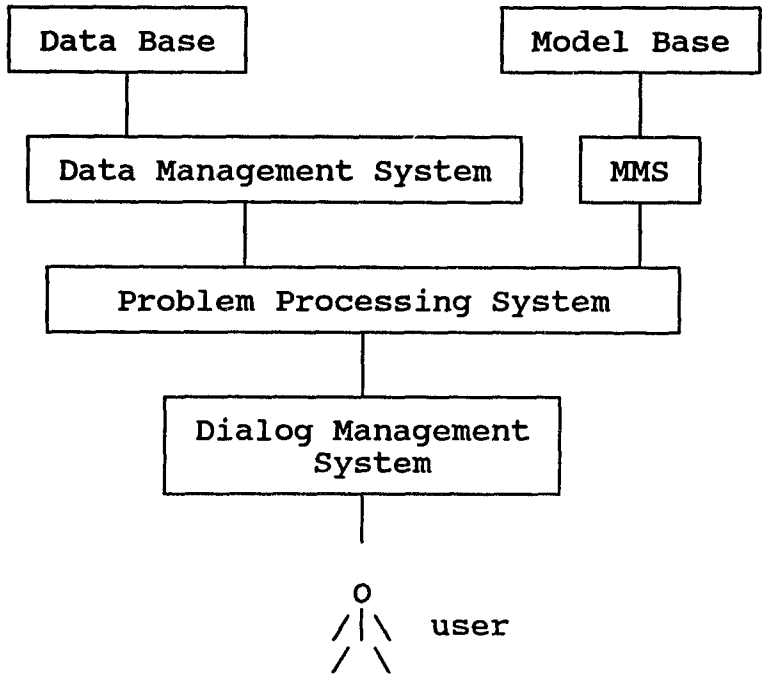


Figure 2a. The Conceptual Architecture of a DSS (Minch and Burns 1983)

The generalized DSS architecture using the functional MMS is reproduced in Figure 2b. In the architecture, a solution procedure library is separated from the model base for the purpose of accomplishing the independence between DSMs and the solution procedures, the third desirable feature of an MMS. A solution procedure library contains a collection of solution algorithms to various DSMs, and is under the management of an independent solution procedure management system. An example of such a system is the Guide of Available Mathematical Software (GAMS) which manages information about large quantities of mathematical and statistical software at the National Bureau of Standards (Boisvert et

al. 1985). The representation and implementation of a solution procedure management system is a different research issue from that of MMSs and is beyond the discussion of the present study (Dolk and Konsynski 1984; Geoffrion 1987).

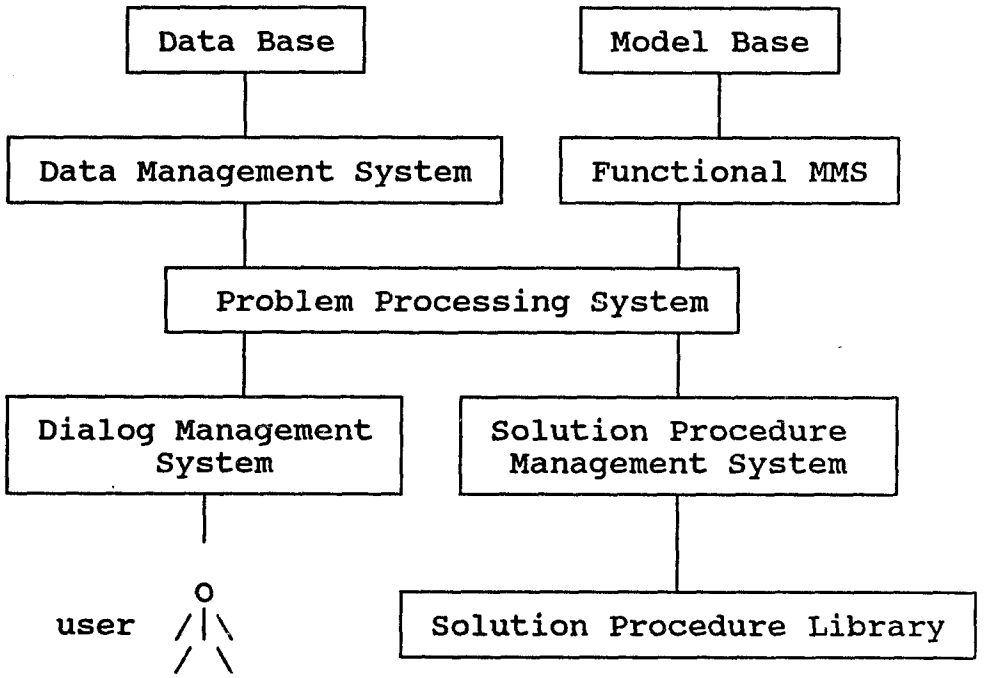


Figure 2b. The Revised Architecture of a DSS

The functional MMS approach, at the macro level, maintains a functional model base which contains a collection of first-order DSM predicates. Each first-order DSM predicate has a unique name and represents an abstract description of a DSM. For each DSM predicate in the model base, a functional data base is maintained to provide the complete description of a DSM. Basically, a functional data base is a definitional system which

expresses the elements of a DSM as relational tables and the mathematical structure as a set of definitions. A functional data base is nothing but a relational data base (Codd 1970) expanded with a set of definitions. The structure of a functional data base is much like that of an active functional system (Risch et al. 1988).

Significance of the Study

The contribution of the present study to the MMS issue can be deliberated at both the macro and micro levels. At the macro level, the foundation of functional MMSs is first-order logic (Robinson 1965; Chang and Lee 1973). The basis on first-order logic implies that the functional MMS is a knowledge-based system. The inclusion of first-order logic can eventually lead to the application of AI techniques to help users select useful DSMs to solve a decision problem.

At the micro level, a functional data base is created and managed based on the relational theory (Codd 1970) which is the mainstream of data base theory so far. The excellent characteristics of relational theory permit the functional MMS to be flexible, to be data and application independent, and to reflect users' views of DSMs. Furthermore, the relational approach not only has powerful support for handling large volumes of data, but also allows a DSM to be drawn directly from a relational data base. As such, an MMS can be closely integrated with

the data management system within a DSS. On top of all these, the relational approach provides users with a uniform set of well-established data manipulation functions to maintain both data of routine use and data referenced in DSMs.

In spite of all the fantastic potentials brought by the foundations of first-order logic and relational theory, the functional MMS is widely applicable within the field of management science/operations research (MS/OR). This is due to the fact that the definitional system expressed by a functional data base is actually the one envisioned at the core of a structured model, which is widely applicable in the field of MS/OR (Geoffrion 1987).

Organization of the Dissertation

This chapter provides an introduction to the present study. The literature of MMS is reviewed in Chapter 2. The chapter also describes the research opportunities revealed by the literature review. Chapter 3 contains basic definitions and notation of the functional MMS approach, presents a procedure to define a functional DSM, and closes with an illustrative example.

After the fundamentals of the functional MMS approach are introduced, Chapter 4 addresses its characteristics from the aspects of DSM representation and manipulation, model base organizations, the desirable features of MMSs and the correspondence to the MMSs for

mathematical programming (MP) DSMs. The uses of the functional MMS are generally discussed in Chapter 5. Chapter 6 discusses the design issues of a functional data base including primitive and virtual relations. Chapter 7 describes an implemented DSM translator which converts an instance of a functional DSM to be in a format which can be directly entered to and solved by a solution procedure. Finally, Chapter 8 presents some closing comments and outlines opportunities for further research.

CHAPTER 2

LITERATURE REVIEW

In the literature of MMSs, many approaches have been proposed to address the MMS problem. These works are first reviewed in the chapter from the aspects of DSM representation (Liang 1988; Shaw, Tu, and De 1988) and manipulation (Dutta and Basu 1984, Shaw, Tu, and De 1988; Bhargava, Bieber and Kimbrough 1988). Then the model base organizations (Bhargava, Bieber, and Kimbrough 1988) of these MMSs are delineated, followed by a discussion of their features and ways of managing DSMs at the macro and micro levels. Next, the chapter describes some MMSs for MP DSMs which are the concentration of the present study. Finally, the literature review is concluded by restating the problem addressed in the present study.

Related MMS Approaches

Briefly speaking, previous MMS approaches represent DSMs utilizing either knowledge representation techniques or relational data base theory. The advantages and disadvantages of these MMS approaches have been summarized by Applegate, Konsynski and Nunamaker (1987).

Knowledge-based Approaches to MMSs

The knowledge-based approaches to MMSs include predicate calculus (Bonczek, Holsapple, and Whinston 1981b), the frame system (Dolk and Konsynski 1984; Fedorowicz and Williams 1986; Applegate, Konsynski, and Nunamaker 1986; Dolk 1986) and the SI-Net, the semantic inheritance network (Elam, Henderson, and Miller 1980; Applegate, Konsynski, and Nunamaker 1986).

Predicate calculus deals with first-order predicates of which the values are either true or false, but not both. A first-order predicate is a declarative sentence with arguments. It is like a function which maps a list of arguments, constants and/or variables, to a truth value. Predicate calculus has powerful search and selection functions for manipulating DSMs at an abstract level, and hence is a great tool for macro-level MMSs. Nevertheless, predicate calculus is precluded from being used at the micro level due to the unsatisfactory performance not only in handling large volumes of data but also in drawing DSMs directly on data from a data base.

The frame system (Minsky 1975) is a structured knowledge representation technique for representing common knowledge about physical objects, locations, situations, and people. It supplies an excellent mechanism for linking user problem descriptions with actual DSM formulations. Yet, as explained in the introduction,

there are difficulties in employing the frame system to solve the MMS problem.

The SI-Net is another structured knowledge representation technique used to solve the MMS problem. A SI-net consist of nodes and links between the nodes. Nodes represent objects, concepts, and events; links represent their interrelations. It is considerably flexible to describe a wide range of DSMs and their interrelations using SI-nets. However, similar to the frame system, the SI-Net has neither design methodology nor evaluation criteria regarding the design of a SI-Net. Furthermore, it is not clear what kind of SI-Nets is required to represent a DSM, and how a SI-Net interfaces with the data management system within a DSS (Dolk and Konsynski 1984).

The Relational Approach to MMSs

A relational MMS (Blanning 1982) is an MMS based on relational data base theory. Being the most commonly used in data base design, the relational approach allows an MMS to be closely integrated with the data management system within a DSS. Moreover, the relational approach provides users with a uniform set of manipulation functions to maintain data of regular use and data referenced in DSMs. Consequently, the relational approach is an excellent candidate for accommodating the numeric components and mathematical structures of DSMs in a model base.

Unfortunately, the proposed relational approach manages DSMs at the macro level, rather than at the micro level. In the relational MMS, a DSM is defined as a computer executable program, and described in a tabular format at the abstract level. The knowledge about DSMs is limited to inputs and outputs, the attributes incorporated in the relational tables. In regard of macro-level model management, the drawback of the relational approach is lack of inferencing capability. It is crucial that an MMS can reason in order to assist users in selecting useful DSMs during decision-making.

Model Base Organization

DSMs must be organized in a model base to facilitate DSM access according to associations among and within DSMs (referred to as **DSM relationships**). DSM relationship can be distinguished as either inter-DSM (i.e., between DSMs) or elemental (i.e., within a DSM). An overview of various types of DSM relationship is presented in Table 2.

Table 2.--Various Types of DSM Relationships

Inter-DSM Relationship	Elemental Relationship
Interface relationship (Blanning 1982) and Structural similarity	Definitional, Bounding, Causal, and Correlational (Paradice and Courtney 1987)

Inter-DSM relationship exists between two or more DSMs. A principal inter-DSM relationship is interface

(i.e., input/output) relationship. Using interface relationship to organize a model base has been a major design feature of the MMSs (Applegate, Konsynski, and Nunamaker 1986) at the macro level. Another important type of inter-DSM relationship is structural similarity. Many DSMs, particularly MP DSMs, are similar in structure; modeling efficiency can be greatly improved if similar DSMs are clustered together within a model base.

Another kind of DSM relationship is elemental relationships which denote the associations between elements of a DSM. Paradice and Courtney (1987) have identified various types of elemental relationships: definitional, bounding, causal, and correlational. Among them, definitional and bounding relationships compose the mathematical structures of DSMs and must be included in the complete descriptions of DSMs. Definitional relationship is the mathematical relationship among the numeric elements, and bounding relationship is the lower and upper limits of the numeric elements.

Theoretically, an MMS can embody any type of DSM relationship in a model base by using special link types. For example, in the SI-Net proposed by Elam, Henderson, and Miller (1980), the DSM relationship "is-a-part-of" is described using a DATTR link which defines the elements of a DSM, and the DSM relationship "are-structured-as" is expressed using a STRUCTURE link which represents the

structure of a DSM (i.e., how the elements are put together).

An overview of the DSM relationship used in the previous MMSs is shown in Table 3. At the macro level, most of the MMSs organize a model base using the interface relationship. Some MMSs also allow utilizing structural similarity through defining special link types. At the micro level, the MMSs use special link types to express the mathematical structure of a DSM. In some MMSs, a DSM is defined as a computer executable program and hence, the mathematical structure of a DSM is not explicitly expressed but buried deeply inside the computer program. Under the circumstance, it is very difficult for a user to comprehend a DSM without studying the computer program thoroughly.

Ideally, an MMS should allow users to access DSMs based on either interface relationship or structural similarity. To allow model access using interface relationship, an MMS simply needs to incorporate inputs and outputs as descriptive attributes in the abstract descriptions of DSMs. Nevertheless, it is not as easy for an MMS to support the access of DSMs according to structural similarity. The reason is that similar DSMs can only be identified by examining the complete descriptions of DSMs. In other words, supplying model access according to structural similarity requires that

elements and mathematical structures of DSMs be extensively expressed and extensively utilized.

Table 3.--DSM Relationships Used in Previous MMSs

MMSs	Inter-DSM Relationship	Elemental Relationship
SI-Net	Any kind (e.g., "are structured as")	Any kind (e.g., "is a part of")
First-order predicate	Interface relationship	-
Relational	Interface relationship	-
Meta- and model abstractions	Interface relationship	Any kind
Connection graphs and frames	Interface relationship	Any kind
SI-Net and frames	Any kind	Any kind

Although not an approach to the MMS problem, structured modeling (Geoffrion 1987) is a framework aiming to represent the semantic and mathematical structure of a formal specification DSM using definitional dependencies. Such an effort provides the foundation for micro-level model management. Structured modeling reflects similar variables and parallel structures of DSMs through the proper use of mnemonic variable names and indices. Besides, it describes DSMs in a dimension-free manner to retain semantics of a DSM and to simplify the

representations of parallel DSM structures.

Features of the Previous MMSs

As mentioned in the introduction, an MMS needs to preserve several features in order to bridge the gap between users and a computer: it needs to be knowledge-based, flexible, independent, and reflecting a user's view of a DSM. The features of the previous MMSs are summarized in Table 4.

Almost all of the proposed MMS approaches are knowledge-based except the relational approach. In addition, to some extent, the majority of the MMSs are flexible in encompassing a wide range of DSMs, supporting DSMs for different application domains, and providing DSMs with data from various sources. However, these MMSs are often dependent on application domains, detailed reference data, and/or solution procedures.

Previous MMSs at the Macro and Micro Levels

The ways that the proposed MMSs dealing with DSMs at the macro and micro levels are presented in Table 5. The development of MMSs has been concentrated on the macro-level MMS in the recent literature (Liang and Jones 1988). As shown on the table, there are three two-level representation schemes: meta and model abstractions (Dolk and Konsynski 1984), connection graphs of frames (Fedorowicz and Williams 1986), and SI-Nets of frames

(Applegate, Konsynski, and Nunamaker 1986).

Table 4.--Features of Previous MMSs

MMSs	Features
SI-Net	Knowledge-based. Flexible in providing multiple DSMs, multiple views of a DSM but not various data sources. Dependent on application domains and/or representation schemes. Reflecting users' view.
First-order predicate	Knowledge-based. Inflexible. Independent of application domains and representation schemes. Not reflecting users' view.
Relational approach	Not knowledge-based. Flexible in providing various data sources but not multiple DSMs and multiple views of a DSM. Independent of application domains and representation schemes. Not reflecting users' view.
Meta- and model abstractions	Knowledge-based. Flexible in providing multiple DSMs but not multiple views of a DSM and various data sources. Dependent on application domains and/or representation schemes. Reflecting users' view.
Connection graphs and frames	Knowledge-based. Inflexible. Dependent on application domains and/or representation schemes. Reflecting users' view.
SI-Net and frames	Knowledge-based. Flexible in providing multiple DSMs, multiple views of a DSM but not various data sources. Dependent on application domains and/or representation schemes. Reflecting users' view.

Table 5.--DSM Representations at both the Macro and Micro Levels

Authors	Macro Descriptions	Micro Descriptions	Detailed Data
Elam, Henderson, Miller (1980)		Model SI-Net of well-formed formulas (wffs).	Values in slots.
Bonczek, Holsapple, Whinston (1981b)	Predicate calculus (wffs).		
Blanning (1982)	Virtual relations.		
Dolk, Konsynski (1984)	Meta-abstractions.	Model Abstractions of wffs.	Values in slots.
Fedorowicz, Williams (1986)	Connection graphs of wffs.	Frame.	Values in slots.
Applegate, Konsynski, Nunamaker (1986)	SI-Net.	Frame.	Values in slots.

As mentioned in the introduction, these three MMS approaches, at the macro level, guide the application of DSMs by applying various knowledge representation techniques to represent and employ knowledge of DSMs. They, at the micro level, unanimously suggest describing the components and mathematical structure of a DSM by using the frame system. Problems of using a frame system to describe a DSM have been delineated.

Meta- and Model Abstractions

A model abstraction describes the elements and structure of a DSM. It is a combination of several knowledge representation schemes, but most resembles the frame system. A model abstraction consists of three sections: data objects, procedures, and assertions. All of the sections are expressed in first-order predicate calculus. The data objects section enumerates the data items and types comprising a DSM. The procedures section lists each procedure with the data objects it accesses and the data objects it returns. The assertions section specifies rules governing a DSM and contains the information about relationships between data objects and procedures.

Since a model abstraction may contain other model abstractions as data objects or procedures, it is very easy to construct aggregate or composite DSMs from existing DSMs. It is also straightforward to build meta-abstractions using other model abstractions as data objects in order to express and detect similarities, differences, and interactions between DSMs.

Connection Graphs of Frames

A connection graph is a first-order logic representation scheme expressed in clause form. In the system proposed by Fedorowicz and Williams (1986), a connection graph is used to implicitly enumerate all the

solution paths using existing DSMs. On a connection graph, eight different literal types can be defined to find values for parameters of DSMs.

A literal is either a first-order predicate or the negation of a first-order predicate. A frame literal is the type of literals which link a connection graph and a model base. Each frame literal represents a reference to a particular DSM. For each DSM, there exists an associated frame which specifies the parameter structure, default parameter values, data input/output characteristics, inherent assumptions, related DSMs, optional DSM configurations, and other specific details for the appropriate application.

SI-Nets of Frames

Applegate, Konsynski, and Nunamaker (1986) suggest using a SI network to link a set of DSMs. Such a SI-Net functions as a classification system of DSMs, and allows for the update, storage and retrieval of DSMs. Each frame on a SI-Net describes the attributes and solution rules for a given class of DSMs and also for specific DSMs within a class.

MMSs for Mathematical Programming DSMs

Supposedly, the discussion of MMSs should not be confined to any particular type of DSMs. However, it is almost impossible to address the issue of representing

and formulating DSMs in isolation from specific types of DSMs, particularly at the micro level. In the present study, the MMS problem is addressed by focusing on deterministic MP DSMs because they are among the most commonly used DSMs in the field of MS/OR.

In practice, there are MMSs developed for supporting use of MP DSMs. Two successfully implemented computer-based systems are PLATOFORM, PLAnning TOol written in dataFORM (Palmer et al. 1984) and CAMP, Computer-Aided Modeling and Planning (Sagie 1986). Based on the model abstraction concept, a prototype MMS--the GXMP (Generalized eXperimental Mathematical Programming) system--is also developed for controlling and managing linear programming (LP) DSMs (Dolk and Konsynski 1984; Dolk 1986).

PLATOFORM

PLATOFORM is a software system which supports more than one hundred MP applications within Exxon. In other words, the fact that PLATOFORM is general enough to support a wide range of MP applications has been justified through the practical experience with the system. PLATOFORM is a combination of two systems: the EMPS (Enhanced Mathematical Programming System) and DATAFORM.

EMPS is an extensive redesign, enhancement, and modification of the standard MP system, MPS/360; whereas DATAFORM, a comprehensive model management language

exploiting the idea of symbolic identifiers. Among other things, the provision for user-defined symbolic identifiers is the most fundamental concept of PLATOFORM which provides users with the capability of defining symbolic (i.e., mnemonic) identifiers, rather than numeric indices, to denote elements of a DSM. To maintain data referenced in MP DSMs, PLATOFORM supports a subset of the full generalized-array model based on set theory and develops its own data base management and manipulation facilities.

CAMP

On the other hand, CAMP is an integrated modeling and planning system which provides a systematic view of the modeling and planning problem to assist planning tasks. A plan built via CAMP is composed of model files, data banks, and other constituents. A model file contains the definition of an MP DSM; a data bank, the definitions and the associated values of variables referenced in the MP DSM.

Similar to PLATOFORM, the most important feature of CAMP is also the exclusive use of symbolic array subscripts to increase legibility of MP DSMs. The data organization supported within CAMP also resembles the one used in PLATOFORM, the generalized-array model.

GXMP

On the basis of model abstraction, GXMP controls and manages a collection of LP DSMs by supporting the mathematical view of LP DSMs. Furthermore, GXMP provides the capability of transforming an LP DSM representation into a matrix format which can be directly input to a solution procedure to have the LP DSM solved. Within GXMP, the components of an LP DSM representation include, among other things, a set of statements written in a modeling language and a network database.

Modeling language used in GXMP bears a lot of resemblance to XML--a hypothetical high-level modeling language (Fourer 1983)--which also uses symbolic array subscripts exclusively. One other major component of GXMP, a network data base, contains data referenced in the LP DSM and are under the control of a data base management system which conforms to a subset of the CODASYL data base standards (CODASYL Systems Committee 1971).

Comparison of PLATOFORM, CAMP and GXMP

By comparing PLATOFORM, CAMP, and GXMP, all these systems provide users with special-purpose high-level modeling languages (MLs) to formulate and enter an MP DSM into a computer. This is however not a coincidence. Fourer (1983) contends that MLs have advantages in the areas of verifiability, modifiability, documentation, independence, and simplicity. Fourer further suggests

that MLs alleviate the specific drawbacks of matrix generators--the traditional approach to input MP DSMs into a computer--in the aspects of data representation, naming components of a DSM, ordering coefficients, and representation of special constraints.

Regarding the maintenance of values and variables referenced in MP DSMs, both PLATOFORM and CAMP use the generalized-array model as the major data organization, and GXMP uses CODASYL network data bases. It is hard to believe that none of the systems employs the rich body of principles and structures provided by the relational theory, the most popular data base technology. This phenomenon can be explained as follows.

On one hand, PLATOFORM and CAMP evolve from standard LP systems which are indeed the implementations of Dantzig's simplex algorithm, the most efficient solution procedure for LP DSMs. As requested by the simplex algorithm, the standard LP systems usually solve an LP DSM by representing it as a coefficient matrix. Hence, instead of a data base, PLATOFORM and CAMP use the generalized-array model (i.e., a generalized version of coefficient matrix) as the major data organization.

On the other hand, GXMP is developed following a parallel yet independent path of the relational data base theory. At the time GXMP was being developed, the relational data base technology was still on its course of

maturity, and the CODASYL network data base was considered as a standard in the field of data base theory. That is why GXMP uses CODASYL network data bases, instead of relational data bases, as the major data organization.

Now that the development of relational data base theory has reached the maturity, it is natural and reasonable to apply the technology in the design of an MMS. The possible argument may be that the generalized-array model allows users to view data as tables which is consistent with the relational data base. Though this is the case, the command syntax of the generalized-array model is quite different from that of relational data base. It is very inconvenient for users to learn and use two distinct set of data manipulation functions in maintaining the data stored in ordinary data bases and the data referenced in MP DSMs. Therefore, it is necessary to explore the possibility of applying relational technology to managing data referenced in a DSM.

The Research Issue

The literature review of MMSs leads to an issue regarding the MMS problem: it is necessary to develop an MMS approach which allows DSMs to be manipulated at both the macro and micro levels, yet preserves the desirable features. The review of MMSs for MP DSMs also reveals the trend of adopting a relational data base as the major data organization. The solution proposed in the present study

is the functional MMS approach. As pointed out in the introduction, the functional approach to MMSs attempts to furnish a two-level MMS which preserve the desirable features. It also aims to provide a unifying design for both the MMS and data management system.

Table 6 provides an overview of the functional MMS approach. The functional MMS represents a DSM as a first-order predicate at the macro level, and as a relational data base with a collection of statements at the micro level. For every DSM, the macro-level description has to be consistent with the detailed expressions and certainly is the abstraction of it.

Table 6.--The Functional Approach to MMS

	At the Macro Level	At the Micro Level
DSM representation	DSM predicates with a unique DSM name, inputs, outputs, objective functions, and constraints.	A functional data base.
DSM manipulation	Formal logic resolution and state-space search.	Relational and domain algebra.
DSM relationships	Interface relationship and structural similarity.	Definitional dependencies and bounding relationship.

At the macro level, the functional model management framework follows the trend of research and employs interface relationship extensively. Furthermore, the functional model management approach permits the

utilization of structural characteristics of DSMs, since the complete descriptions of DSMs are available in the associated data base. DSMs are represented as first-order predicates which have powerful macro-level model manipulation functions. Based on interface or structural similarity, DSMs can be created, selected, retrieved, modified and integrated as individual entities.

The functional MMS expresses the mathematical structure of a DSM as a definitional system using the relational technology. The relational approach allows DSMs to draw directly from a relational data base. It also provides users with a uniform set of well-established data manipulation functions. Hence, users do not need to learn two different languages to manipulate data in the ordinary data bases and data referenced in DSMs. Furthermore, the excellent features of a relational data base permit the functional MMS to preserve most of the desirable features.

Nonetheless, the expressive capacity of an ordinary relational data base is not sufficient enough to describe the mathematical structure of a DSM. To overcome such a limitation, the functional MMS expands a relational data base with an intensive part which uses operations of domain algebra (Merrett 1984) to express mathematical relationships on attributes. After the elements and mathematical structure of a DSM are defined, a DSM can be

combined with distinct detailed data to formulate similar DSM instances.

Side-effect-free functions have been introduced to avoid computational redundancy, and to allow sharing of computational models (Orman 1986). Nevertheless, a computational model is mainly concerned with the algorithmic details of computations. Such a computational model does not reflect reality, and hence is not a DSM. Rather, it is a solution procedure. According to Geoffrion (1987), DSMs are users of solution procedures, and the representation and implementation of solution procedures is a different research issue. Consequently, Orman's work is totally different from the functional MMS approach proposed in the present study. For DSMs, the semantic structures are much more important than their algorithmic aspects.

CHAPTER 3

BASICS OF THE FUNCTIONAL MMS

In this chapter, an introduction is given to the definitions and notations of the functional MMS, followed by definitions and syntax of domain algebra operations--the important foundation for expressing the mathematical structure of a DSM. Then the procedures of defining a DSM are delineated, succeeded by an illustrative example based on the transportation DSM, a classic MS/OR application.

The functional MMS approach requires that users have some training in building a DSM, and be capable of defining the numeric elements and mathematical structure of a DSM. This requirement is justifiable because an MMS is to support, rather than to replace users in formulating DSMs. It is believed that the current AI technology is still in its course of development, and cannot perfectly reproduce an expert's behavior of constructing and verifying DSMs using a computer (Elam and Konsynski 1987). Furthermore, a user is more willing to accept and implement the result suggested by the solution of a DSM which is explored, developed and validated by him/herself (Elam and Konsynski 1987). Table 7 presents the complete description of a functional DSM.

Table 7.--The Complete Description of a Functional DSM

The Macro-level Description: A DSM Predicate

DSM (DSM_Name, Input[1], Output[2], Objective[3], Constraint[4])

[1] Input is a list of primitive numeric attributes which are defined in the intensive data base, and of which the values are given in an extensive data base.
[2] Output is a list of variable attributes which are defined in the intensive data base; the values of the variable attributes are to be determined by the solution of a DSM.
[3] Objective is a list of single-valued virtual numeric attributes whose optimal values are to be determined by the DSM.
[4] Constraint is a logical attribute which defines all kinds of constraints of the DSM.

The Micro-level Representation: A Functional Data Base

I. The Intensive Data Base:
containing a collection of primitive relations and statements. Each primitive relation is composed of a primitive numeric attribute and the identifier. Each statement defines a virtual numeric attribute using operations of domain algebra, or a logical attribute using logical expressions in conjunctive normal form.

II. The Extensive Data Base:
containing a set of rows for each primitive relation in the intensive data base; these sets of rows consist of data values of the parameter attributes.

Definitions and Notations

The functional MMS approach expresses a DSM as a set of mathematical associations (such as equations, inequalities, etc.) among numeric attributes of entity and relationship types. The term **entity**, serving as an abstraction for objects, is commonly used and generally

understood in data base design. The concept of entity and relationship types of a DSM originates from the popular entity-relationship (E-R) model (Chen 1976) of data base design. Thus, entity and relationship types of a DSM can be identified using Chen's E-R model, and expressed on an entity-relationship diagram (ERD). Since the mathematical associations of a DSM correspond to some down-to-earth connections in the real world, it makes sense to capture the semantic structure of a DSM by examining associations between numeric attributes of the entity and relationship types of a DSM.

Entity Type

An **entity** of a DSM is an abstract but meaningful thing involved in a DSM. For example, factories and customers are the entities involved in the transportation DSM. Entities are grouped into entity types in a meaningful and natural way. An **entity type** corresponds to a group of entities which are alike in nature. Entities of the same type are different from each other only in the detailed data. For example, FACTORY and CUSTOMER are two entity types directly resulting from grouping the entities of the transportation DSM, factories and customers. Different entities of the same type are referred to as **instances** of the entity type.

Primitive Symbolic Attribute

A primitive symbolic attribute is a property of an entity type which uniquely identifies an entity and distinguishes it from other entities of the same type. It is called primitive because it bears a value which uniquely identifies an entity and cannot be computed. A primitive symbolic attribute is the primary key of a relation in a relational data base. In other words, entities of the same type are treated as named symbols according to the values taken by the primitive symbolic attribute. For example, locations of factories and customers can be used as identifiers of factories and customers, respectively. The methodology of choosing an appropriate primitive symbolic attribute for an entity type can be found in the literature of data base theory (Teorey, Yang, and Fry 1986).

Each primitive symbolic attribute should be given a mnemonic name to capture the meaning of the attribute. For example, the primitive symbolic attributes of factories and customers can be called "FactoryLoc" and "CustomerLoc", respectively since they stand for the locations of factories and customers. Because symbolic attributes are always primitive, they can be simply referred to as **symbolic attributes.** However, a primitive symbolic attribute (e.g., telephone number) may have numeric contents. Such an attribute is still called

"symbolic" because the value is for the purpose of identifying entities, not for direct computation.

Relationship Type

Relationships are meaningful interactions between the entities. Depending on the entity types joining the interactions, relationships can also be clustered into different types. Hence, a **relationship type** refers to the similar interactions between entity types. A relationship is usually identified by the data values that identify the entities participating in the relationship. Hence, most often a relationship type has a composite identifier which is the combination of the symbolic attributes of the entity types taking part in the relationships.

For example, in the transportation DSM, the shipments of merchandise from factories to customers are the interactions between factories and customers. Because the interactions are between entity types FACTORY and CUSTOMER, they are considered of the same type (called SHIPMENT) and are uniquely identified by a pair of factory and customer locations (i.e., "FactoryLoc", "CustomerLoc").

Primitive Numeric Attribute

A primitive numeric attribute is a measurable property of an entity or relationship type. It is called primitive because its values can be observed and is

collectable. For examples, the monthly requirement ("Demand") is a primitive numeric attribute of CUSTOMER; whereas the monthly production capacity ("Supply"), a primitive numeric attribute of FACTORY. The relationship type SHIPMENT also has a primitive numeric attribute, that is the unit transportation cost ("Unit_Trans_Cost") from a factory to a customer.

For every possible value of a symbolic attribute, the primitive numeric attribute can take only one value. In other words, a primitive numeric attribute is functionally dependent on the identifier of an entity or relationship type. Hence, the identifier of an entity or relationship type is also the determinant of the corresponding primitive numeric attribute.

Constants referenced in a DSM can be defined as **constant attributes** which are identifier-free primitive numeric attributes. One example is the hourly rate of setting up a particular machine, "Setup_Hour_Rate", a constant in the classic economic order quantity (EOQ) DSM with multiple (independent) items. "Setup_Hour_Rate" can take only one single value and does not need an identifier.

The values of primitive numeric attributes may be known or unknown to the users. When the values are known to the users, the primitive numeric attributes represent the input parameters of a DSM, and can be referred to as

parameter attributes. The actual number of inputs required to solve a DSM is determined by the numbers of different values taken by the identifiers of all the parameter attributes.

When the values are unknown to the users, a primitive numeric attribute represents unknown decision variables which are to be determined by the solution of a DSM. Thus, it can be referred to as a **variable attribute.** The values of a variable attribute are computed by taking the whole DSM into consideration. They are usually solved by an iterated algorithm. In other words, more often there does not exist a closed-form formula which can express the mathematical relationship between a variable attribute and other numeric attributes. An example of a variable attribute is the numeric attribute "Ship_Qty" of SHIPMENT of which the values are to be determined by solving the transportation DSM.

Variable attributes play a very important role in the functional MMS. In fact, it is the existence of variable attributes that distinguishes relational tables used in a functional data base from those used in an ordinary relational data base. Since the values of variable attributes are to be determined, it seems improper to classify variable attributes as "primitive" numeric attributes. However, when variable attributes cannot be solved by a closed-form formula, they are still

classified as "primitive" because (1) they can be defined and manipulated the same way as parameter attributes, and (2) their dimensions (i.e., the actual number of decision variables) can also be specified the same way as those of parameter attributes.

Primitive Relation

A **primitive relation** is the set of a primitive numeric attribute and its identifier. Each primitive relation contains information about either an entity type (called an **entity relation**) or a relationship type (called a **relationship relation**). Four primitive relations of the transportation DSM defined so far are as follows.

FACTORY (FactoryLoc, Supply)
CUSTOMER (CustomerLoc, Demand)
SHIP_QTY (FactoryLoc, CustomerLoc, Ship_Qty)
UNIT_TRANS_COST
(FactoryLoc, CustomerLoc, Unit_Trans_Cost)

Following the convention of the relational data base, the primitive relations are capitalized and identifiers are underlined. Rows (called tuples in relational data base) of primitive relations contain data values referenced in a DSM except those of variable attributes. The dimension of a primitive relation refers to the number of rows containing data values for that relation. For example, the dimension of the primitive relation FACTORY is the number of factories; the dimension of CUSTOMER is the number of customers. A primitive

relation made up of a constant attribute (called **constant relation**) is identifier-free, and is defined in a reduced format which does not have an identifier.

Virtual Numeric Attribute

A **virtual numeric attribute** is also a measurable property of an entity or relationship type. Different from a primitive numeric attribute, a virtual numeric attribute is defined as a mathematical expression of other numeric attributes, primitive or virtual. In other words, the values of a virtual numeric attribute are derivable (i.e., computable) from the values of other numeric attribute and not directly observable.

The numeric attributes used in the mathematical expression to define a virtual numeric attribute are called the **operand attributes** of the virtual numeric attribute. The relations containing the operand attributes are the **operand relations**. In other words, the values of a virtual numeric attribute are computed from the values of their operand attributes. For example, since the transportation cost from a factory to a customer is computed as the product of the unit transportation cost and the shipment quantity, the attribute "Trans_Cost" is a virtual numeric attribute and defined as Unit_Trans_Cost * Ship_Qty. Both "Unit_Trans_Cost" and "Ship_Qty" are primitive numeric attributes of the relationship type SHIPMENT and have a composite identifier, ("FactoryLoc",

"CustomerLoc").

Not all the attributes can be used as operand attributes, especially virtual attributes. The defined virtual attribute itself cannot be one of the operand attributes. Furthermore, any virtual attribute computed from the defined virtual attribute cannot be an operand attribute. In other words, neither direct nor indirect recursive definition is allowed. It is possible that a variable attribute be defined as a virtual numeric attribute. The situation happens when there exists a closed-form formula which can express the mathematical relationship between a variable attribute and the parameter attributes directly.

Virtual Relation

A **virtual relation** is the set of a virtual numeric attribute and the identifier. It is implicitly specified by the mathematical definition of the virtual numeric attribute. For example, the definition of "Trans_Cost" (i.e., Unit_Trans_Cost * Ship_Qty) implicitly defines a virtual relation, TRANS_COST, as follows.

TRANS_COST (FactoryLoc, CustomerLoc, Trans_Cost).

The identifier of a virtual relation is implicated by the identifiers of operand relations. In the example, because "FactoryLoc" and "CustomerLoc" together form the composite identifier of both operand relations (UNIT_TRANS_COST and

SHIP_QTY), it is then the identifier of the virtual relation TRANS_COST. Since tuples of a virtual relation are derived from tuples of operand relations, the dimension of a virtual relation is completely determined by the dimensions of operand relations. Hence, using the single expression, Trans_Cost = Unit_Trans_Cost * Ship_Qty, the virtual relation TRANS_COST is completely defined and the data values are derivable from the values of the relations UNIT_TRANS_COST and SHIP_QTY. Such an expression is **dimension-independent** because it represents the similar relationships among the values of "Trans_Cost", "Ship_Qty", and "Unit_Trans_Cost" for all the shipments from factories to customers.

Virtual Logical Attribute

A **virtual logical attribute** defines a collection of comparisons between numeric attributes, primitive or virtual. The basic format of defining a virtual logical attribute is as follows.

<vla> = <cnf>

where <vla> denotes a virtual logical attribute and <cnf> is a logical expression in conjunctive normal form (i.e., conditions are connected with the logical operator, AND). The formal syntax for a logical expression is as follows.

```
<cnf>          ::= (<condition>) AND <cnf> |
                   (<condition>)
<condition>    ::= <na> <binary op> <na>
<binary op>    ::=  < | ≤ | = | > | ≥
<na>           ::= <pna> | <vna>
```

where <pna> is a primitive numeric attribute and <vna> is a virtual numeric attribute. The value of a virtual logical attribute can only be either true or false; but not both.

The two numeric attributes in a condition must be comparable. In other words, the attributes compared within a condition must be of identical dimension (i.e., they need to have the same identifier). Usually, one of the two numerical attributes in comparison involves variable attributes directly or indirectly. In other words, the truth value of a virtual logical attributes is always unknown and dependent on the solution of a DSM. This is because that each condition of a virtual logical attribute specifies a type of constraints for a DSM. A virtual logical attribute is usually used to set up all the constraints under which a DSM is to be solved. Since the truth value of a virtual logical attribute is to be determined, the virtual logical attribute can be simply referred to as logical attributes without causing any ambiguity.

For example, a logical attribute "Meet" can be defined as follows to describe all the constraints of the transportation DSM.

Meet = (Qty_Supplied ≤ Supply) AND
(Qty_Received = Demand)

Both virtual numeric attributes "Qty_Supplied" and "Qty_Received" are subtotals of the variable attribute "Ship_Qty". The attribute "Qty_Supplied" denotes the total quantity shipped from each factory, while the attribute "Qty_Received" means total quantity received by each customer.

Qty_Supplied ≤ Supply, the first condition of "Meet" denotes the type of constraints that all factories cannot ship to customers more than they can produce (i.e., their monthly capacities). The actual number of constraints of this type is dependent on the dimensions of the relations QTY_SUPPLIED and SUPPLY. Similarly, the second condition, Qty_Received = Demand, denotes another type of constraints that customers receive the same quantities as their monthly requirements. As a result, the logical attribute "Meet" defines all the constraints of the transportation DSM. The transportation DSM must be solved to satisfy all the constraints defined in "Meet".

Intensive, Extensive and Functional Data Base

An **intensive data base** (IDB) defines the numeric elements and mathematical structure of a DSM. It contains a set of primitive relations and a collection of statements. Some primitive relations define the data

referenced in a DSM using parameter attributes; some define decision variables of a DSM using variable attributes. This part of an IDB contains the intensions of a relational data base which includes at least one variable attribute.

Another part of an IDB is composed of a set of statements for defining virtual numeric and logical attributes. These statements are independent of the dimensions of parameter and variable attributes. The reason is that the dimensions of primitive relations are the actual numbers of rows stored in the primitive relations; while the dimensions of virtual relations are implied by those of their operand relations. Hence, there is no need of using indices to define virtual numeric and logic attributes.

It is in an **extensive data base** (EDB) where data values of parameter attributes are maintained together with values of their identifiers. These sets of rows are the input data values required to solve a DSM. For each variable attribute defined in an IDB, there is also a set of dummy rows which specify the actual number of decision variables of that type. An EDB contains the extensions of a relational data base. The data stored in an EDB can be maintained using the well-established manipulation functions of relational data base.

The dimension of a DSM is usually implied by the

dimensions of the primitive relations. Particularly, the dimension of an MP DSM tends to be multiplicative functions of the dimensions of entity relations. An EDB and IDB together compose a **functional data base** which contains information about the numeric elements, mathematical structure and detailed data of a DSM, and describes a DSM at the micro level.

DSM Predicate

A **DSM predicate** is a 5-place first-order predicate which provides an abstract description for a DSM.

DSM (DSM_Name, Input, Output, Objectives, Constraint)

The predicate name is "DSM". The first argument is a unique mnemonic name of a DSM. The second argument denotes a list of input parameter attributes. The values of parameter attributes must be given in order to formulate an instance of a DSM. The third argument "Output" represents a list of variable attributes and their values are to be determined by the solution of a DSM.

The fourth argument "Objectives" of the predicate DSM is the counterpart of objective functions in an optimization problem. If a DSM is not an optimization problem, the associated "Objectives" is an empty list. In the case of an optimization DSM, each attribute in the list is preceded by an 'max' or 'min' to indicate

maximization or minimization. Attributes in the list "Objectives" (called **objective attributes**) are usually virtual, and have variable attributes as operand attributes. They are, most of the time, reduced to single values (i.e., identifier-free). Finally, the fifth argument "Constraint" is a logical attribute which sets up the constraints of a DSM.

When a DSM is used in another DSM as a submodel, its DSM predicate can be embedded in an IDB to define virtual numeric attributes. Attributes in the embedded DSM predicate are bound to the attributes of the DSM predicate stored in the functional model base according to their relative positions.

For example, suppose that the model base has a DSM predicate for the transportation DSM,

```
DSM (Transport, [Unit_Trans_Cost, Supply, Demand],
[Ship_Qty], [(min Total_Trans_Cost)], Meet).
```

When the following DSM predicate is embedded in an IDB,

```
DSM (Transport, [Unit_Trans_Cost, Capacity,
Requirement], [Units], [(min Total_Trans_Cost)],
Meet).
```

the variable attribute "Units" are bound to the attribute "Ship_Qty" in the original DSM since they are in the relative positions of the two DSM predicates. The values of "Units" can be obtained by solving the submodel after the values of the parameter attributes "Capacity" and

"Requirement" are passed to the submodel.

Functional DSM and Functional Model Base

A DSM predicate together with the corresponding functional data base is called a **functional DSM** which contains the complete descriptions of DSMs. A DSM predicate and the corresponding IDB define the DSM schema of a class of DSMs. The DSM schema can be combined with different EDBs to formulate similar DSM instances which, after converted to a form, can be input directly to a solution procedure for a solution. A **functional model base** contains a collection of functional DSMs. The DSMs in a functional model base can be of repeated use when their predicates being embedded in the IDBs of other DSMs.

Operations of Domain Algebra

The descriptive power of the functional MMS hinges on the types of operators available to define virtual numeric attributes. The more operator types are for defining virtual numeric attributes, the more DSM types can be described and incorporated in a functional model base. Operators used to define virtual numeric attributes in the functional MMS are based on operations of domain algebra, an algebra of attributes.

In a relational data base, domain algebra provides a formalism for expressing mathematic operations on attributes (Merrett 1984). The operations of domain

algebra are consistent with the operations provided by relational algebra for relations in a relational data base. Domain algebra consists of a set of operations. Most operators used to define virtual numeric attributes are directly transcribed from the ideas introduced by Merrett (1984). They are modified to incorporate the operations of relational algebra and to allow the transformations of numeric relations. By using operations of relational and domain algebra together, relations can be manipulated in both horizontal and vertical directions.

There are three sets of operations used to define virtual numeric attributes (Table 8): scalar operation, reduction, and functional mapping. There is one kind of scalar operation and two kinds of reduction and functional mapping; which are simple versus equivalence reductions, and simple versus partial functional mapping.

Scalar Operation

A scalar operation defines a virtual numeric attribute in terms of an arithmetic expression on operand attributes. The arithmetic operations are applied repeatedly to tuples of the operand relations to compute values of the virtual attribute. The format of defining a virtual numeric attribute using a scalar operation is as below.

Table 8.--Operations of Defining Virtual Numeric Attributes in an IDB

Operations of Defining Virtual Numeric Attributes	Format
1. Scalar operation.	<expression>
2. Simple reduction.	<op> of (<na>)
3. Equivalence reduction.	<op> of (<na>) by (<ca>)
4. Simple functional mapping.	<op> of (<na>) order (<oa>)
5. Partial functional mapping.	<op> of (<na>) order (<oa>) by (<ca>)

<vna> = <expression>

where <vna> denotes a virtual numeric attribute. The formal syntax of a scalar expression is as follows.

```
<expression> ::= <expression> + <term> |
                 <expression> - <term> | <term>
<term>       ::= <term> * <factor> |
                 <term> / <factor> | <factor>
<factor>     ::= <primary>
<primary>    ::= - <primary> | <element>
<element>    ::= (<expression>) | <na> | #<na>
<na>         ::= <pna> | <vna>
```

where #<na> is the dimension of a numeric attribute <na>, and <pna> denotes a primitive numeric attribute.

A virtual attribute defined by a scalar operation has at least two operand relations. Tuples of operand relations need to be combined into a single relation before the computations can take place. There are three situations when operand relations are combined: they are

of identical dimension, they are of similar dimensions, or they have disjoint identifiers.

Operand relations are of identical dimensions if and only if they have an identical identifier. An example is computing the transportation cost from a factory to a customer ("Trans_Cost") by multiplying unit transportation cost ("Unit_Trans_Cost") by the shipment quantity ("Ship_Qty"). Since both operand relations UNIT_TRANS_COST and SHIP_QTY use the composite identifier ("FactoryLoc", "CustomerLoc"), they are of identical dimension. The resulting virtual relation TRANS_COST also uses the composite identifier ("FactoryLoc", "CustomerLoc"). In this case, the virtual attribute "Trans_Cost" retains the dimensions of "Unit_Trans_Cost" and "Ship_Qty". Therefore, if operand relations are of identical dimension, the result of a scalar operation retains their dimensions.

Operand relations are of similar dimensions if their identifiers are overlapping (i.e., they share at least one common symbolic attribute in their identifiers). In this situation, operand relations need to be transformed to compatible forms (i.e., to have identical identifiers) before the computations can take place. The transformations are done by applying the JOIN operation of relational algebra over the common symbolic attributes of their identifiers.

For example, suppose that the relation UNIT_PRICE (FactoryLoc, Unit_Price) contains the information about the unit price of the product from a factory. The revenue of a factory from each customer by supplying the product can be computed as follows.

Factory_Revenue = Unit_Price * Ship_Qty

In this case, the operand attributes "Unit_Price" and "Ship_Qty" share "FactoryLoc" in their identifiers and are of similar dimensions. To obtain compatible operand attributes, a temporary relation (TEMP) is formed by joining relations UNIT_PRICE and SHIP_QTY over the common attribute "FactoryLoc".

TEMP = JOIN UNIT_PRICE, SHIP_QTY OVER FactoryLoc
TEMP (FactoryLoc, CustomerLoc, Qty1, Unit_Price1)

After the transformation, the temporary operand attributes "Qty1" and "Unit_Price1" become compatible and their data values are used to compute values of the virtual attribute "Factory_Revenue". Thus, when a virtual attribute is defined by a scalar operation on two operand relations of similar dimensions, the resulting virtual attribute retains the larger dimension of the operand relations, and is uniquely identified by the union of their identifiers.

When operand relations of a scalar operation have disjoint identifiers, they are transformed by taking the Cartesian product. That means that tuples of the

temporary relation are generated by combining each row of an operand relation with every row of another operand relation. The dimension of the resulting virtual attribute is then the product of the dimensions of the operand relations.

For example, to compute the amount each customer pays every factory by purchasing the product, the attribute "Amount" is computed as follows.

Amount = Unit_Price * Demand

The operand attributes "Unit_Price" and "Demand" use disjoint identifiers, "FactoryLoc" and "CustomerLoc", respectively. To compute "Amount", a temporary relation (TEMP2) is formed by taking the Cartesian product of the relations UNIT_PRICE and DEMAND.

TEMP2 = PRODUCT UNIT_PRICE, DEMAND
TEMP2 (FactoryLoc, CustomerLoc, Qty2, Demand2)

Attributes "Qty2" and "Demand2" become compatible, and the result of the scalar operation is as follows.

AMOUNT (FactoryLoc, CustomerLoc, Amount)

Hence, when the operand relations of a scalar operation have disjoint identifiers, the dimension of the resulting virtual relation is the product of the dimensions of the operand relations, and the identifier is the union of their identifiers.

Simple Reduction

A virtual attribute defined by a simple reduction has only one operand relation. A simple reduction applies an associative and commutative operator to all the values of an operand attribute and reduces them to a single value. The basic format of using a simple reduction is as below.

<vna> = <op> of (<na>)

where <op> is an associative and commutative operator and <na> is the operand attribute. Examples of associative and commutative operations include addition (+), multiplication (*) and maximum (max). Since the result of a simple reduction is always a single value, there is no need to determine the identifier of the resulting virtual relation. For example, the total transportation cost can be defined as the sum of the transportation costs ("Trans_Cost").

Total_Trans_Cost = + of (Trans_Cost)

Equivalence Reduction

Equivalence reduction is similar to a simple reduction except it produces different results (such as subtotals) for different groups of tuples in the operand relation. Each group of tuples is characterized by the

same value for the specified control attributes. Therefore, equivalence reduction involves an operand attribute and a set of control attributes. The computation is conducted on the values of the operand attribute within a group which bears the same values for the control attributes. The basic format of an equivalence reduction is as follows.

<vna> = <op> of (<na>) by (<ca>)

where <op> is an associative and commutative arithmetic operation, <na> is the operand attribute and <ca> is a list of control attributes.

For example, the total shipment quantity from a factory ("Qty_Supplied") can be computed by adding up the shipment quantities to all the customers from the factory. The addition is performed within a group of tuples which have an identical value for the attribute "FactoryLoc".

Qty_Supplied = + of (Ship_Qty) by (FactoryLoc)

The resulting virtual relation is as follows.

QTY_SUPPLIED (FactoryLoc, Qty_Supplied).

Notice that the attribute "FactoryLoc" is part of the identifier of the relation SHIPMENT and it becomes the identifier of the resulting virtual relation QTY_SUPPLIED. In general, control attributes in an equivalence reduction

are a proper subset of the identifier of the operand relation and become the identifier of the resulting virtual relation. The dimension of the result of an equivalence reduction is the number of different values taken by the control attributes.

Simple Functional Mapping Operation

In a relational data base, order of tuples in a relation is insignificant. However, in a functional data base, the order of tuples is sometimes important. It is possible that a numeric identifier (e.g., time periods) specifies an order which is used to compute values of a virtual numeric attribute. In other words, there may be some sort of functional associations among values of a numeric attribute. For example, the production of a time period may be dependent on the production of the previous time period.

Simple functional mapping provides a means for handling the association among tuples of a relation. Using a simple functional mapping operation, values of the resulting virtual numerical attribute are dependent on the order of tuples in an operand relation. It is the ordering attributes that specify the order of tuples in an operand relation. An expression using a simple functional mapping is as follows.

<op> of (<na>) order (<oa>)

where <op> is an operator, <na> is the operand attribute and <oa> is a set of ordering attributes. Examples of the operators include addition (+), predecessor (pre) and successor (succ). The operand attribute must be functionally dependent on the ordering attributes. For example, cumulative annual sales can be defined as the sum of annual sales based on the order determined by values of the numeric identifier "Year".

Cum_Sales = + of (Sales) order (Year)

Suppose that the annual sales stored in the primitive relation SALES are as follows.

SALES (Year, Sales)

Year	Sales
1984	150,000
1985	175,000
1986	200,000
1987	210,000
1988	225,000

The tuples of the virtual relation CUM_SALES are then as follows.

CUM_SALES (Year, Cum_Sales)

Year	Cum_Sales
1984	150,000
1985	325,000
1986	525,000
1987	735,000
1988	960,000

Notice that the cumulative annual sales for the year 1986 is computed by adding up the annual sales for the years 1984, 1985, and 1986.

Another example demonstrates the use of the predecessor operator (pre) to define the previous annual sales (Pre_Sales).

Pre_Sales = pre of (Sales) order (Year)

According to the definition, the virtual relation PRE_SALES becomes the following.

PRE_SALES (Year, Pre_Sales)

Year	Pre_Sales
1984	225,000
1985	150,000
1986	175,000
1987	200,000
1988	210,000

Note that the predecessor operator is cyclic; the previous annual sale for the year 1984 is the annual sale of 1988. As can been observed from both examples, the result of a simple functional mapping is of the dimension of the operand relation.

Partial Functional Mapping Operation

Partial functional mapping operation extends functional mapping operation the same way that equivalence reduction extends simple reduction. It produces different results (e.g., divisional cumulative sales) for groups of tuples classified by the specified control attributes. The basic format of a partial functional mapping operation is as follows.

<vna> = <op> of (<na>) order (<oa>) by (<ca>)

where <op> is an operator, <na> is the operand attribute, <oa> is a list of ordering attributes, and <ca> is a set of control attributes. The operand attribute must be functionally dependent on both ordering and control attributes. For example, cumulative annual sales for different divisions can be computed using a partial functional mapping operation.

```
Div_Cum_Sales = + of (Div_Sales) order (Year)
                  by (Division)
```

Assume that the annual sales for different divisions are as follows.

DIV_SALES (Division, Year, Div_Sales)

Division	Year	Div_Sales
A	1986	80,000
A	1987	120,000
A	1988	115,000
B	1986	60,000
B	1987	90,000
B	1988	100,000

The virtual relation DIV_CUM_SALES is then as below.

DIV_CUM_SALES (Division, Year, Div_Cum_Sales)

Division	Year	Div_Cum_Sales
A	1986	80,000
A	1987	200,000
A	1988	315,000
B	1986	60,000
B	1987	150,000
B	1988	250,000

Similar to functional mapping, the result of a partial functional mapping operation retains the dimension of the operand relation and uses the combination of the ordering

and control attributes as the composite identifier.

The Procedure of Defining a Functional DSM

In the functional MMS, the procedure for formulating a DSM consists of the following steps.

Step 1. Identify entities of the DSM, classify the entities into entity types, and determine the identifier of each entity type. Give each entity type and its identifier mnemonic names. Also specify numeric attributes of entity types referenced in the DSM.

Step 2. Identify all the relationship types among the entity types specified in Step 1. Combine the identifers of the participating entity types to form the identifiers of the relationship types. Given each relationship type a meaningful name. Also specify numeric attributes of relationship types referenced in the DSM.

Step 3. Draw an entity-relationship diagram (ERD) to display the relationship between the entity and relationship types identified in the previous two steps. Convert each entity or relationship type into an intermediate relation which includes the identifier and the related numeric attributes referenced in the DSM.

Step 4. Decompose intermediate relations into elementary form which contains only one numeric attribute and the identifier. After the decomposition, each numeric attribute should appear only in one elementary relation. Each elementary relation also contains one numeric

attribute only, and thus can be named after the numeric attribute. Following the convention of relational data base, identifiers of elementary relations are underlined and names of elementary relations are capitalized.

Step 5. Distinguish virtual (i.e., computable) relations from primitive relations. Construct the IDB by collecting primitive relations together.

Define each virtual numeric attribute explicitly using scalar operations, reductions, functional mapping, or other DSM predicates. Define also constant attributes for the constants used in the definitions of virtual numeric attributes. Expand the IDB by including definitions of virtual numeric attributes.

Step 6. Consider the objective functions of the DSM. Define a virtual attribute with a mnemonic name to represent each objective function. Repeat defining any operand attribute that is not a variable attribute and whose data values are not directly available. Definitions of the objective functions and the related virtual attributes are included in the IDB.

Step 7. Consider different types of constraints of the DSM. For each type of constraints, form a condition. A condition specifies a binary relation between two comparable numeric attributes; each of which must be given a meaningful name and defined appropriately. Repeat defining any operand attribute that is not a variable

attribute and whose data values are not directly available. Definitions of the related virtual attributes are collected in the IDB.

Define a logical attribute which is the conjunction of all the conditions. The definition of the logical attribute is also included in the IDB.

Step 8. Construct a DSM predicate for the DSM by determining the mnemonic DSM name, parameter attributes, variable attributes, objective functions and constraints of the DSM. A DSM name has to be unique in the functional model base. Parameter attributes are the attributes of which the data values are directly available. They must be defined as primitive relations in the IDB. Variable attributes represents different kinds of the decision variables whose values are to be determined by the solution of the DSM. An objective function is usually denoted by a single-valued virtual numeric attribute and defined in the IDB. Constraints of the DSM are expressed as the logical attribute defined in Step 7.

An Illustrative Example

The procedure of specifying a functional DSM is illustrated using the transportation DSM. The transportation DSM is a classic MS/OR application which is to determine the optimal shipment quantity from a factory to a customer such that the total transportation cost is minimal.

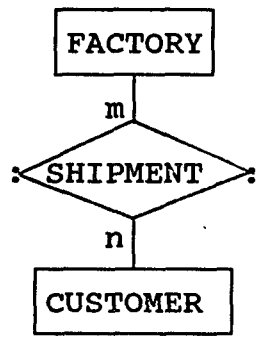
Step 1. It is obvious that factories and customers are the entities of the transportation DSM. They can be naturally classified into two entity types, FACTORY and CUSTOMER. Entities of FACTORY are identified by their locations, so are entities of CUSTOMER. The identifiers can be defined as "FactoryLoc" and "CustomerLoc", respectively.

The entity type FACTORY should contain the numeric attribute "Supply" which denotes the monthly capacity of a factory. Similarly, the entity type CUSTOMER should contain the numeric attribute "Demand" which denotes the monthly requirement of a customer.

Step 2. Shipments are the physical interactions between factories and customers. The relationship type is called SHIPMENT and can be identified uniquely by a pair of factory and customer locations, "FactoryLoc" and "CustomerLoc". The relationship type SHIPMENT contains two numeric attributes, "Ship_Qty" and "Unit_Trans_Cost". The attribute "Ship_Qty" represents the shipment quantity from a factory to a customer which is the decision variables of the transportation DSM. The attribute "Unit_Trans_Cost" represents the unit transportation cost from a factory to a customer.

Step 3. Figure 3 shows the ERD for the transportation DSM. It is straightforward to convert the entity types FACTORY, CUSTOMER and the relationship type

SHIPMENT into three intermediate relations: FACTORY, CUSTOMER and SHIPMENT. The relations FACTORY and CUSTOMER represent entity types and are called **entity relations;** the relation SHIPMENT is from a relationship type and is called a **relationship relation.** The cardinality of the relationship type SHIPMENT is many-to-many since a factory can deliver the merchandise to several customers and a customer can receive the merchandise from several factories.



FACTORY (FactoryLoc, Supply)

SHIPMENT (FactoryLoc, CustomerLoc, Ship_Qty, Unit_Trans_Cost)

CUSTOMER (CustomerLoc, Demand)

Figure 3. The ERD for the Transportation DSM

Step 4. The intermediate relations are transformed into the following elementary relations.

```
SUPPLY     (FactoryLoc, Supply)
SHIP_QTY   (FactoryLoc, CustomerLoc, Ship_Qty)
UNIT_TRANS_COST
           (FactoryLoc, CustomerLoc, Unit_Trans_Cost)
DEMAND     (CustomerLoc, Demand)
```

Step 5. The IDB contains the elementary relations obtained in Step 4.

Step 6. The objective function of the

transportation DSM is to minimize the total transportation cost which can be defined as below.

Total_Trans_Cost = + of (Trans_Cost)

The values of the operand attribute "Trans_Cost" are not directly available, and must be defined.

Trans_Cost = (Unit_Trans_Cost) * (Ship_Qty)

Both operand attributes, "Unit_Trans_Cost" and "Ship_Qty", are defined in the IDB.

Step 7. There are two types of constraints corresponding to the viewpoints of the suppliers factories and the customers. From a supplier's viewpoint, no factories can ship to customers more than what they can produce every month. From a customer's point of view, quantity received by a customer must meet his monthly need. The constraints can be defined as follows.

Meet = (Qty_Supplied ≤ Supply) AND
(Qty_Received = Demand)

This definition of the logical attribute "Meet" is also incorporated into the IDB.

The attribute "Qty_Supplied" denotes the total shipment quantity from a factory, and can be computed by adding up the shipment quantities to all the customers from the factory. Similarly, the attribute "Qty_Received" denotes the total quantities received by a customer, and

can be computed by adding up the shipment quantities from all the factories to the customer. In other words, both "Qty_Supplied" and "Qty_Received" are aggregate functions of "Ship_Qty". The definitions of attributes "Qty_Supplied" and "Qty_Received" are as follows.

```
Qty_Supplied = + of (Ship_Qty) by (FactoryLoc)
Qty_Received = + of (Ship_Qty) by (CustomerLoc)
```

"Supply" and "Demand" are numeric attributes of the entity types "Factory" and "Customer" and their data are directly available. Note that "Qty_Supplied" and "Supply" are comparable because they both use "FactoryLoc" as their identifers; so are "Qty_Received" and "Demand".

Step 8. Finally, the transportation DSM can be described at the abstract level as a DSM predicate.

```
DSM (Transport, [Unit_Trans_Cost, Supply, Demand],
        [Ship_Qty], [min Total_Trans_Cost], Meet)
```

"Transport" is the name of the transportation DSM. There are three parameter attributes: "Supply", "Demand" and "Unit_Trans_Cost". Their dimensions depend on the data values stored in an EDB. "Ship_Qty" is the variable attribute and represents the decision variables to be determined by the solution of the transportation DSM. The objective of the transportation DSM is to minimize (min) the total transportation cost ("Total_Trans_Cost") under the condition that the logical attribute "Meet" is

satisfied.

Figure 4 presents the general structure of the transportation DSM using algebraic expressions. The complete functional description of the transportation DSM is summarized in Table 9. The DSM predicate and the IDB together define the general structure of the transportation DSM. Attributes of the transportation DSM are summarized in Table 10.

$$\text{Minimize} \quad \sum_{i=1}^{m} \sum_{j=1}^{n} COST_{ij} \, QTY_{ij}$$

$$\text{Subject to} \quad \sum_{j=1}^{n} QTY_{ij} \leq SUPPLY_i \quad \text{for } i=1 \text{ to } m$$

$$\sum_{i=1}^{m} QTY_{ij} = DEMAND_j \quad \text{for } j=1 \text{ to } n$$

$$QTY_{ij} \geq 0$$

where $COST_{ij}$ denotes unit transportation cost from factory i to customer j,
QTY_{ij} denotes shipment quantities from factory i to customer j,
$SUPPLY_i$ is the monthly capacity of factory i,
$DEMAND_j$ is the monthly requirement of customer j,
m is the number of factories and
n is the number of customers.

Figure 4. The General Structure of the Transportation DSM

Table 9.--The Complete Functional Description of the Transportation DSM

The Macro-level Description: A DSM Predicate

```
DSM (Transport, [Unit_Trans_Cost, Supply, Demand],
       [Ship_Qty], [(min Total_Trans_Cost)], Meet)
```

Note: The transportation DSM (Transport) is to determine shipment quantities (Ship_Qty) from factories to customers such that total transportation cost (Total_Trans_Cost) is minimized (min) under the constraints that Meet is true given unit transportation costs from factories to customers (Unit_Trans_Cost), monthly capacities of factories (Supply), and monthly requirements of customers (Demand).

The Micro-level Representation: A Functional Data Base

I. The IDB: a collection of primitive relations, and a set of definitions for virtual attributes.

```
UNIT_TRANS_COST (FactoryLoc, CustomerLoc,
                 Unit_Trans_Cost)
SUPPLY    (FactoryLoc, Supply)
DEMAND    (CustomerLoc, Demand)
SHIP_QTY (FactoryLoc, CustomerLoc, Ship_Qty)
Total_Trans_Cost = + of (Trans_Cost)
Trans_Cost       = (Unit_Trans_Cost) * (Ship_Qty)
Qty_Supplied     = + of (Ship_Qty) by (FactoryLoc)
Qty_Received     = + of (Ship_Qty) by (CustomerLoc)
Meet = (Qty_Supplied ≤ Supply) AND
       (Qty_Received = Demand)
```

II. The EDB: see Figure 5a.

Table 10.--Attributes of the Transportation DSM

Attributes	Type	Interpretations
*** Symbolic attributes:**		
FactoryLoc	character	location of a factory
CustomerLoc	character	location of a customer
*** Parameter attributes:**		
Unit_Trans_Cost	numeric	unit transportation cost from a factory to a customer
Supply	numeric	supply of a factory
Demand	numeric	demand of a customer
*** Variable attributes:**		
Ship_Qty	numeric	shipment quantity from a factory to a customer
*** Virtual attributes:**		
Total_Trans_Cost	numeric	total transportation cost
Trans_Cost	numeric	transportation cost from a factory to a customer
Qty_Supplied	numeric	quantity supplied by a factory
Qty_Received	numeric	quantity received by a customer
Meet	logical	constraints of meeting demands and supplies

An instance of the transportation DSM can be easily formulated by submitting data values of the parameter attributes in an EDB. An example is shown in Figure 5a. Note that there are five dummy rows included in the EDB for relation SHIP_QTY even though it contains the variable attribute "Ship_Qty". This is necessary because that the route from Chicago to Atlanta for some reason is not possible. Thus, dummy rows of relation SHIP_QTY need to be included in the EDB to allow the possible decision variables to be defined.

UNIT_TRANS_COST

(FactoryLoc,	CustomerLoc,	Unit_Trans_Cost)
Dallas	Pittsburgh	23.50
Dallas	Atlanta	17.75
Dallas	Cleveland	32.45
Chicago	Pittsburgh	7.60
Chicago	Cleveland	25.75

SUPPLY (FactoryLoc, Supply)

Dallas	20,000
Chicago	42,000

DEMAND (CustomerLoc, Demand)

Pittsburgh	25,000
Atlanta	15,000
Cleveland	22,000

SHIP_QTY

(FactoryLoc,	CustomerLoc,	Ship_Qty)	Corresponding Variable Names
Dallas	Pittsburgh	-	Qty_{11}
Dallas	Atlanta	-	Qty_{12}
Dallas	Cleveland	-	Qty_{13}
Chicago	Pittsburgh	-	Qty_{21}
Chicago	Cleveland	-	Qty_{23}

Figure 5a. An EDB of the Transportation DSM

The corresponding LP formulation is presented in Figure 5b. Another DSM instance of the transportation DSM can be obtained by changing the tuples of the primitive relations. Many "what-if" questions can also be asked by modifying data values of the parameter attributes.

$$\text{Minimize } 23.50\,QTY_{11} + 17.75\,QTY_{12} + 32.45\,QTY_{13} + 7.60\,QTY_{21} + 25.75\,QTY_{23}$$

$$\begin{aligned} \text{Subject to } & QTY_{11} + QTY_{12} + QTY_{13} \le 20{,}000 \\ & QTY_{21} + QTY_{23} \le 42{,}000 \\ & QTY_{11} + QTY_{21} = 25{,}000 \\ & QTY_{12} = 15{,}000 \\ & QTY_{13} + QTY_{23} = 22{,}000 \\ & QTY_{11}, QTY_{12}, QTY_{13}, QTY_{21}, QTY_{23} \ge 0 \end{aligned}$$

Figure 5b. The LP Formulation for the Transportation DSM in Figure 5a.

CHAPTER 4

CHARACTERISTICS OF THE FUNCTIONAL MMS

The functional MMS provides a two-level conceptual framework to couple numeric and symbolic knowledge of DSMs. The functional MMS allows use of mathematical structures, in addition to the interface relationship, of DSMs to select a useful DSM. Furthermore, it has a uniform design for data of routine use and for the data referenced in DSMs. In fact, the functional MMS has a number of excellent characteristics. The characteristics of functional MMSs are addressed from the aspects of DSM representation and manipulation, model base organization, desirable features of MMSs, and the correspondence to an ML, MAGIC.

DSM Representation and Manipulation

The way that a DSM is represented and manipulated using the functional approach can be deliberated at both the macro and micro levels (Table 11). In the functional model base, a DSM is represented as a first-order predicate. DSMs, represented as first-order predicates, can be dealt with by using the predicate calculus which has powerful search and selection functions for

manipulating DSMs at an abstract level.

Table 11.--The Functional Approach to MMSs

	At the Macro Level	At the Micro Level	Detailed Data
DSM representation	DSM predicates with a unique DSM name, inputs, outputs, objective functions, and constraints.	An IDB.	EDBs.
DSM manipulation	Formal logic resolution and state-space search.	Relational and Domain Algebra.	Operations of a relational language

Moreover, the functional MMS uses an IDB to describe the numeric components and mathematical relationships of a DSM. The numeric components of a DSM are described as a set of primitive relations and the mathematical structure as a set of dimension-free definitions. The basis of an IDB is on relational data base theory. As to the detailed data of a functional DSM, they are maintained in an EDB using operations of a relational language.

Macro-Level DSM Representation

A DSM predicate is a declarative sentence with five arguments: the DSM name, inputs, outputs, objectives, and constraints. It maps the list of a DSM name, inputs, outputs, objectives, and constraints to a truth value, either true or false. Basically, a DSM predicate declares a class of DSMs about its name, inputs, outputs, objective

functions, and constraints. For example, the DSM predicate

```
DSM (Transport, [Unit_Trans_Cost, Supply, Demand],
    [Ship_Qty], [(min Total_Trans_Cost)], Meet)
```

declares that there exists a class of DSMs "Transport" which require parameter attributes, "Unit_Trans_Cost", "Supply", and "Demand" to determine "Ship_Qty" such that "Total_Trans_Cost" is minimized (min) when conditions specified in "Meet" are met.

Macro-Level DSM Manipulation

The foundation of the functional MMS at the macro level is on first-order logic (Robinson 1965; Chang and Lee 1973). The logic-based DSM predicates can be dealt with by the predicate calculus, a system of characterizing deductions. The "calculus" in "the predicate calculus" means that it gives a way of determining whether statements are true. As such, the predicate calculus consists of a language for expressing statements, and rules for inferring new facts from those already known. The predicate calculus enables us to speak of classes of DSMs, to postulate relationships between these DSMs, and to generalize the relationships over classes of DSMs.

DSM selection is a very important DSM manipulation function. DSMs can be selected according to DSM predicates or DSM predicates as well as the associated

functional data bases. Since a DSM predicate contains five arguments, DSMs can be selected based on either of these descriptors at the macro level under the universal relation assumption of relational data base (i.e., each attribute name has a global meaning).

For example, utilizing inputs and outputs of DSM predicates, the functional MMS allows DSM selection based on interface relationship. Suppose that there exists a functional model base which includes the DSM predicates shown in Figure 6.

```
DSM (EOQ, [Demand_Rate, Hold_Cost, Fixed_Cost],
    [Order_Qty], [(min Total_Cost)], -).
DSM (Demand, [Prod_Demand, Bill_Of_Material],
    [Demand_Rate], [], -).
DSM (Hold_Cost, [OC_Rate, Direct_Material_Cost,
    Direct_Labor_Cost, Storage_Cost], [Hold_Cost],
    [], -).
DSM (Fixed_Cost, [Setup_Rate, Setup_Hours,
    Fixed_Material_Cost], [Fixed_Cost], [], -).
```

Figure 6. Some DSM Predicates in A Functional Model Base

The rule of finding the predecessors of the DSM "EOQ" can be stated as below.

> A DSM is a predecessor of "EOQ" if the output of the DSM contains any parameter attribute of "EOQ".

Under the universal relation assumption, the rule of finding predecessors of "EOQ" can be expressed as the PROLOG statements presented in Figure 7. PROLOG is the best-known of the logic programming languages. In PROLOG,

a predicate or constant name starts with a lower-case letter and a variable name starts with an upper-case character. Constants can also be put in quotes. The symbol ":-" stands for "if".

```
predecessor (DSM, "EOQ") :-                    line 0
    dsm ("EOQ", X, _, _, _),                   line 1
    dsm (DSM, _, Y, _, _),                     line 2
    not (DSM = "EOQ"),                         line 3
    interact (X, Y).                           line 4
```

Figure 7. PROLOG Statements of Finding the Predecessors of "EOQ" DSM

The collection of PROLOG statements reads that a DSM is a predecessor of "EOQ" (line 0) if (:-)

the input list of "EOQ" is X (line 1),
the output list of the DSM is Y (line 2),
the DSM is not "EOQ" (line 3), and
some common attributes appears in both lists X and Y (line 4).

According to the information incorporated in DSM predicates, conflicting DSMs can also be detected. Two DSMs are conflicting with each other if their objective functions are contradictory. In addition, the functional MMS also permits DSM selection utilizing structural characteristics of DSMs. However, the utilization of DSM structural characteristics is not as straightforward as that of the interface relationship because DSM structures are described in the associated IDB, not in DSM predicates. In other words, manipulating DSMs based on

their structural characteristics concerns the macro- and micro-level DSM representation.

Micro-level DSM Representation

For each DSM predicate in the model base, the functional MMS uses a functional data base to provide the complete description and detailed data of the DSM. A functional data base contains an IDB and EDB. Basically, an IDB is a definitional system which expresses the numeric elements of a DSM as relational tables, and mathematical relationship as a set of definitions. On the other hand, an EDB is a collection of tuples providing detailed data for the parameter attributes defined in an IDB. In other words, data referenced in a functional DSM are maintained in a relational data base.

The tabular structure of relations can be readily explained to DSM users. For example, it is easier to explain to a user the meaning of $23.50 in the first row of the primitive relation UNIT_TRANS_COST in Figure 5a than to interpret $23.50, the first coefficient of the objective function in Figure 5b. It is obvious that, in the first case, $23.50 is the unit transportation cost from Dallas to Pittsburgh.

The definitional system expressed in a functional data base bears a lot of resemblance to the one envisioned at the core of a structured model (Geoffrion 1987). The similarity between the two can be illustrated using

feedmix DSM. The class of feedmix DSM is to determine the amount of each material to be blended for animal feeds which reach minimum daily levels of nutrients at the minimal cost of materials. The general structure of feedmix DSM is presented in Figure 8.

$$\text{Minimize} \quad \sum_{i=1}^{m} \text{UNIT\_COST}_i * \text{QTY}_i$$

Subject to

$$\sum_{i=1}^{m} \text{ANALYSIS}_{ij} * \text{QTY}_i \geq \text{MIN\_NUTR}_j \quad \text{for } j=1 \text{ to } n$$

$$\text{QTY}_i \geq 0 \quad \text{for } i=1 \text{ to } m$$

where UNIT_COST_i denotes unit cost of material i
QTY_i is the quantity of material i,
ANALYSIS_{ij} denotes analysis of nutrient j in material i,
MIN_NUTR_j is the minimal requirement of nutrient j,
m is the total number of materials and
n is the total number of nutrients.

Figure 8. The General Structure of Feedmix DSM

Figures 9 and 10, excerpts from Geoffrion's paper, show the genus graph and text-based schema of the structured model, feedmix DSM. Some attributes are renamed to make it easy to compare the structured model with the functional DSM. The genus graph aims to capture the mathematical dependencies among attributes (i.e., natural familial groupings of the numeric components) of a DSM in a dimension-free manner. The text-based schema

describes the general structure of feedmix DSM.

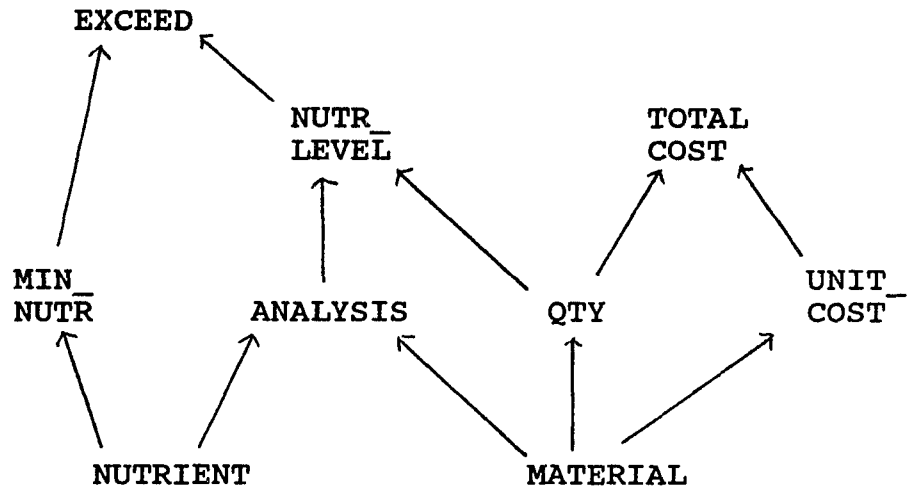


Figure 9. The Genus Graph for Feedmix DSM (Geoffrion 1987)

```
&NUT_DATA NUTRIENT DATA
 NUTRIENTj /pe/
 MIN_NUTR (NUTRIENTj) /a/ {NUTRIENT}: R+

&MATERIALS MATERIALS DATA
 MATERIALi /pe/
 UNIT_COST (MATERIALi) /a/ {MATERIAL}: R
 ANALYSIS (MATERIALi, NUTRIENTj) /a/
    {MATERIAL} x {NUTRIENT}: R+

QTY (MATERIALi) /va/ {MATERIAL}: R+

NUTR_LEVEL (ANALYSIS.j, QTY) /f/ {NUTRIENT};
   SUMi (ANALYSISij * QTYi)

EXCEED (NUTR_LEVELj, MIN_NUTRj) /t/ {NUTRIENT};
   NUTR_LEVELj ≥ MIN_NUTRj

TOTAL_COST (UNIT_COST, QTY) /f/; SUMi (UNIT_COSTi*QTYi)
```

Figure 10. The Text-based Schema for Feedmix DSM Without the Interpretation Part (Geoffrion 1987)

In a structured model, there are five types of elements: primitive entity (pe), compound entity (ce), attribute (a), function (f), and test (t) elements. Figure 11, also an excerpt from Geoffrion (1987), contains a sample of elemental detail tables for feedmix DSM. The skeletal structure of elemental detail tables is determined from the text-based schema.

NUTR

NUTRIENT	INTERP	MIN_NUTR
P	Protein	16
C	Calcium	4

MATERIAL

MATERIAL	INTERP	UNIT_COST
std	standard	1.20
add	additive	3.00

ANALYSIS

NUTRIENT	MATERIAL	ANALYSIS
P	standard	4
P	additive	14
C	standard	2
C	additive	1

QTY

MATERIAL	QTY
std	2.00
add	.50

NUTR_LEVEL

NUTRIENT	NUTR_LEVEL	EXCEED
P	15.00	FALSE
C	4.50	TRUE

TOTAL_COST

TOTAL_COST
3.90

Figure 11. Sample Elemental Detail for Feedmix DSM (Geoffrion 1987)

The ERD of feedmix DSM is presented in Figure 12.

The ERD is different from the genus graph in Figure 9. First, an ERD depicts relationships between related entity types of a DSM, not between attributes. An ERD is to help recognize identifiers of entity and/or relationship types in order to handle subscripts of similar decision variables as well as parameters. Second, an ERD contains only primitive attributes of a DSM; not virtual attributes. This is because a class of DSMs is characterized by the exact mathematical relationships, rather than the dependencies, between primitive/virtual attributes.

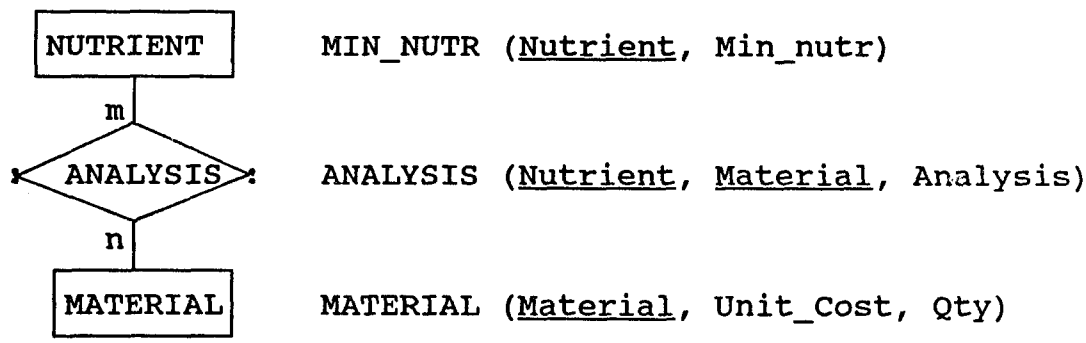


Figure 12. The ERD for Feedmix DSM

The functional MMS represents the exact mathematical relationships in an IDB. The functional description and attributes of feedmix DSM are presented in Tables 12 and 13, respectively. In fact, a genus graph can be produced using the contents of an IDB (Figure 13). Such a genus graph is very similar to the one in Figure 9 except that the entity types (i.e., nutrients and materials) are no

longer included.

Table 12.--The Functional Description of Feedmix DSM

The Macro-level Description: A DSM Predicate

```
DSM (Feedmix, [Unit_Cost, Analysis, Min_Nutr],
       [Qty], [min Total_Cost],  Exceed)
```

The Micro-level Representation: A Functional Data Base

I. The IDB:

```
UNIT_COST (Material, Unit_Cost)
ANALYSIS (Nutrient, Material, Analysis)
MIN_NUTR (Nutrient, Min_Nutr)
QTY (Material, Qty)
Total_Cost    = + of (Material_Cost)
Material_Cost = (Unit_Cost) * (Qty)
Nutr_level    = + of (Matr_Analysis) by (Nutrient)
Matr_Analysis = (Qty) * (Analysis)
Exceed        = (Nutr_level ≥ Min_Nutr) AND
                (Qty ≥ 0)
```

II. The EDB:

UNIT_COST (Material, Unit_Cost)

Material	Unit_Cost
standard	1.20
additive	3.00

ANALYSIS (Nutrient, Material, Analysis)

Nutrient	Material	Analysis
Protein	standard	4
Protein	additive	14
Calcium	standard	2
Calcium	additive	1

MIN_NUTR (Nutrient, Min_Nutr)

Nutrient	Min_Nutr
Protein	16
Calcium	4

QTY (Material, Qty)

Material	Qty	Corresponding Variable Names
standard	-	Qty_1
additive	-	Qty_2

Table 13.--Attributes of Feedmix DSM

Attributes	Type	Interpretations
*** symbolic attributes:**		
Nutrient	symbolic	name of each nutrient
Material	symbolic	name of each material
*** parameter attributes:**		
Unit_Cost	numeric	unit cost of each material
Analysis	numeric	unit of each nutrient in one unit of each material
Min_Nutr	numeric	minimum requirement of each nutrient
*** variable attribute:**		
Qty	numeric	quantity of each material
*** virtual attributes:**		
Total_Cost	numeric	total material cost
Material_Cost	numeric	cost of each material
Nutr_level	numeric	level of each nutrient in each material
Matr_Analysis	numeric	units of each nutrient in each material
Exceed	logical	constraints of exceeding nutrient levels

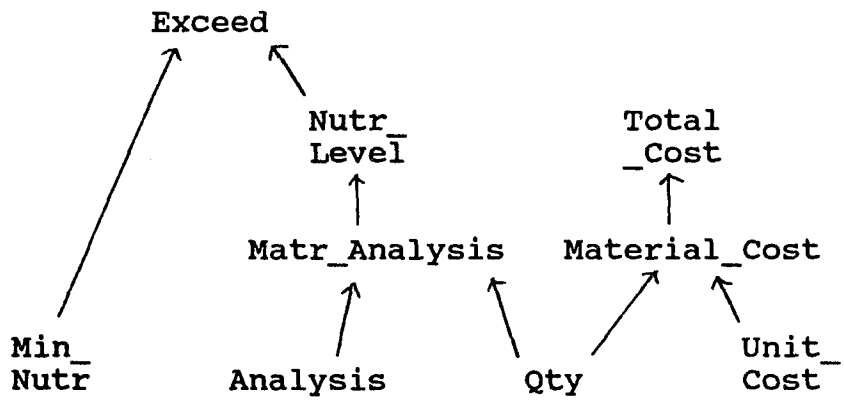


Figure 13. A Genus Graph Generated from the IDB of Feedmix DSM

Though simpler, the IDB of a functional DSM is actually equivalent to the text-based schema of a

structured model. In an IDB, both primitive and compound entities (pe and ce) are expressed as entity/relationship relations, and attributes (a) as numeric properties of entity/relationship relations. Since attributes are tied to entity/relationship relations, their dimensions are determined by identifiers of entity/relationship relations, and there is no need to use index set statements. As to elements of function (f) and test (t), they are defined as virtual attributes using index-free expressions and their dimensions are implied by the dimensions of their operand relations.

As to an EDB, it is obviously similar to the elemental detail tables in Figure 11 except that the virtual relations are not included because their values can be computed from those of their operand relations. An EDB is superior to a set of elemental detail tables since it is a relational data base.

Micro-level DSM Manipulation

The basis of a relational data base allows data of a functional DSM to be maintained using a relational language. Hence, users can use a uniform set of data manipulation functions to manipulate data of routine use and data referenced in DSMs. For example, the unit transportation cost from Dallas to Pittsburgh can be easily changed from $23.75 to $24.75 using the update statements in SQL--Structured Query Language--shown in

Figure 14. SQL is the best known relational data base language (Lans 1988).

```
UPDATE UNIT_TRANS_COST
SET    Unit_Trans_Cost = 24.75
WHERE  FactoryLoc = "Dallas" AND
       CustomerLoc = "Pittsburgh"
```

Figure 14. SQL Update Statements

The definitional system in an IDB provides a natural way of developing a DSM in a hierarchical manner, a time-honored concept that has been used with success to undertake great complexity. A DSM can be developed hierarchically through introducing virtual numeric attributes to capture semantics of mathematical expressions.

For example, in Table 9, the objective of the transportation DSM is to minimize the total transportation cost. The meaning of the objective function is more easily understood through the mnemonic name, Total_Trans_Cost, than the expression to compute the total transportation cost, i.e., $\Sigma\ \Sigma\ COST_{ij} * QTY_{ij}$.

Defining a virtual attribute in terms of other virtual attributes repeatedly represents a series of stepwise refinements based on a hierarchical view of the decision problem. For example, "Total_Trans_Cost" is defined as + of Trans_Cost, i.e., the sum of all the transportation costs. Through the use of the virtual

numeric attribute "Trans_Cost", it is easier for a user to perceive that each term of the objective function represents the transportation cost from a factory to a customer. The process of defining virtual attributes must be repeated until the data of all operand relations are readily obtainable from the EDB.

Model Base Organization

As mentioned in the literature review, DSMs must be organized in a model base to facilitate DSM access according to DSM relationships. Table 14 presents the DSM relationships which can be used in the functional MMS. The utilization of interface relationship is a major macro-level DSM manipulation function and has been discussed earlier. The structural similarity between DSMs can be identified by comparing the definitional and bounding relationships composing mathematical structures of DSMs.

Table 14.--DSM Relationships Can be Used in a Functional Model Base

Inter-DSM Relationship	Elemental Relationship
Interface Relationship	Definitional
Structural Similarity	Bounding

Several inter-DSM relationships can be identified through examining the mathematical structures of DSMs. Before DSMs can be compared, there must be a way of

determining equivalent numeric attributes. A theory of attribute equivalence is published in the literature of data base theory (Larson, Navathe, and Elmasri 1989). Among the several kinds of attribute equivalences, the strong β equality is most useful regarding comparisons of DSMs. Roughly speaking, two attributes are strongly β equal if, for every point in time,

1. there exists a one to one correspondence (the mapping) between their domains;
2. each allowable operation on one attribute has an equivalent allowable operation on another attribute;
3. each semantic integrity constraint of one attribute should be implied by the corresponding semantic integrity constraint of another attribute under the mapping and its inverse;
4. all state change constraints hold under the mapping and its inverse;
5. all security constraints hold under the mapping and its inverse;
6. the mapping and its inverse preserve functional dependencies; and
7. the mapping and its inverse preserve unique identifiers.

Identity

The simplest inter-DSM relationship which can be identified through examining DSM predicates and their mathematical structures is **identity**. Two identical DSMs basically represent the same class of DSMs except attribute names may be different. Given two DSM predicates,

DSM (M_1, I_1, O_1, Obj_1, C_1) and
DSM (M_2, I_2, O_2, Obj_2, C_2)

they are identical (M_1=M_2) if all the arguments are identical, i.e., I_1=I_2, O_1=O_2, Obj_1=Obj_2, and C_1=C_2.

Two input lists are identical (I_1=I_2) if each parameter attribute in I_1 is strongly β equal to the corresponding parameter attribute in I_2. Since all parameter attributes are primitive, determining their strong β equality is simply an application of attribute equivalence theory (Larson, Navathe, and Elmasri 1989). However, because parameter attributes are mainly for reference, conditions (1), (2), (3), (6) and (7) are sufficient to determine their strong β equality.

Two output lists are identical (O_1=O_2) if each variable attribute in O_1 is strongly β equal to the corresponding variable attribute in O_2. The strong β equality of primitive variable attributes can be determined in a way similar to that of parameter attributes. However, because primitive variable attributes are unknown, they are strongly β equal as long as their identifiers are strongly β equal. Deciding whether virtual variable attributes are strongly β equal is a little more complicated. Two virtual attributes are strongly β equal if they are defined on strongly β equal operand attributes using an identical operation.

Two objective lists are identical (Obj_1=Obj_2) if each objective in Obj_1 is identical to the corresponding objective in Obj_2. Two objectives are identical if they

optimize strongly β equal objective attributes. The strong β equality of objective attributes can be determined in a way similar to that of virtual variable attributes. In other words, two strongly β equal objective attributes must be defined on strongly β equal operand attributes using an identical operation.

Finally, two virtual logical attributes are identical ($C_1=C_2$) if each condition in C_1 is identical to the corresponding condition in C_2. Two conditions are identical if they specify an identical binary relationship between two attributes, each attribute in one condition is strongly β equal to the counterpart in another condition.

Equivalence

Another useful inter-DSM relationship which can be identified through examining DSM predicates and their mathematical structures is **equivalence**. Given two DSM predicates for DSMs M_1 and M_2,

DSM (M_1, I_1, O_1, Obj_1, C_1) and
DSM (M_2, I_2, O_2, Obj_2, C_2)

they are equivalent ($M_1 \equiv M_2$) if all the arguments are equivalent (i.e., $I_1 \equiv I_2$, $O_1 \equiv O_2$, $Obj_1 \equiv Obj_2$, and $C_1 \equiv C_2$). Equivalence is less restrictive than strong β equality. In other words, strongly β equality implies equivalency, but not vice versa.

Equivalent DSMs result from equivalent virtual

numeric attributes. Equivalence of two primitive numeric attributes is not different from their strong β equality. However, that is not the case for virtual numeric attributes. Two virtual numeric attributes are equivalent if their definitions are equivalent. For example, in the product mix DSM, also a typical MS/OR application, "Total_Profit" can be defined by either of the two expressions shown in Figure 15. Expression (1) in Figure 15 is equivalent to expression (2).

Since parameter attributes are primitive, determining their equivalence ($I_1 \equiv I_2$) is the same as determining their strong β equality. Two output lists are equivalent ($O_1 \equiv O_2$) if each primitive variable attribute in O_1 is strongly β equal to the corresponding primitive variable attribute in O_2, and each virtual variable attribute in O_1 is equivalent to the corresponding virtual variable attribute in O_2.

Two objective lists are equivalent ($Obj_1 \equiv Obj_2$) if each objective in Obj_1 is equivalent to the corresponding objective in Obj_2. Two objectives are equivalent if they optimize two equivalent objective attributes. Finally, two virtual logical attributes are equivalent ($C_1 \equiv C_2$) if each condition in C_1 is equivalent to the corresponding condition in C_2. Two conditions are equivalent if they specify an identical binary relationship between two attributes, each attribute in one condition is equivalent

to the counterpart of another condition.

$$
\begin{aligned}
&\text{Total\_Profit} \\
&= \text{Total\_Revenue} - \text{Total\_Cost} \\
&= \sum_{i=1}^{m} \text{Revenue}_i - \sum_{i=1}^{m}\sum_{j=1}^{n} \text{Cost}_{ij} \\
&= \sum_{i=1}^{m} (\text{Unit\_Price}_i * \text{Units}_i) - \\
&\quad \sum_{i=1}^{m}\sum_{j=1}^{n} (\text{Resource\_Unit\_Cost}_j * \text{Use}_{ij}) \\
&= \sum_{i=1}^{m} (\text{Unit\_Price}_i * \text{Units}_i) - \\
&\quad \sum_{i=1}^{m}\sum_{j=1}^{n} (\text{Resource\_Unit\_Cost}_j * \text{Unit\_Use}_{ij} * \text{Units}_i)
\end{aligned}
\tag{1}
$$

$$
\begin{aligned}
&\text{Total\_Profit} \\
&= \sum_{i=1}^{m} \text{Profit}_i \\
&= \sum_{i=1}^{m} \{\text{Unit\_Profit}_i * \text{Units}_i\} \\
&= \sum_{i=1}^{m} \{[\text{Unit\_Price}_i - \text{Unit\_Product\_Cost}_i] * \text{Units}_i\} \\
&= \sum_{i=1}^{m} \{[\text{Unit\_Price}_i - \sum_{j=1}^{n} \text{Unit\_Use\_Cost}_{ij}] * \text{Units}_i\} \\
&= \sum_{i=1}^{m} \{ [\text{Unit\_Price}_i - \\
&\quad \sum_{j=1}^{n} (\text{Resource\_Unit\_Cost}_j * \text{Unit\_Use}_{ij})] * \text{Units}_i\}
\end{aligned}
\tag{2}
$$

where m is total number of products, and
n is total number of resources.

Figure 15. Equivalent Definitions of "Total_Profit"

Since expressions (1) and (2) in Figure 15 are equivalent to each other, the functional DSMs of product mix DSM using these two definitions are equivalent (Tables 15 and 16).

Table 15.--The Functional Description of Product Mix DSM Using Definition (1) in Figure 15 for "Total_Profit"

The Macro-level Description: A DSM Predicate
DSM (Prodmix, [Unit_Price, Resource_Unit_Cost, Unit_Use, Resource_Available], [Prod_Units], [(max Total_Profit)], Enough)
The Micro-level Representation: A Functional Data Base
I. The IDB: UNIT_PRICE (Product, Unit_Price) RESOURCE_UNIT_COST (Resource, Resource_Unit_Cost) UNIT_USE (Product, Resource, Unit_Use) RESOURCE_AVAILABLE (Resource, Resource_Available) PROD_UNITS (Product, Prod_Units) Total_Profit = (Total_Revenue) - (Total_Cost) Total_Revenue = + of (Revenue) Revenue = (Unit_Price) * (Prod_Units) Total_Cost = + of (Cost) Cost = (Resource_Unit_Cost) * (Use) Resource_Use = + of (Use) by (Resource) Use = (Unit_Use) * (Prod_Units) Enough = (Resource_Use ≤ Resource_Available) AND (Prod_Units ≥ 0)
II. The EDB: omitted.

Table 16.--The Functional Description of Product Mix DSM Using Definition (2) in Figure 15 for "Total_Profit"

The Macro-level Description: A DSM Predicate
DSM (Prodmix, [Unit_Price, Resource_Unit_Cost, Unit_Use, Resource_Available], [Prod_Units], [(max Total_Profit)], Enough)

The Micro-level Representation: A Functional Data Base
I. The IDB: UNIT_PRICE (Product, Unit_Price) RESOURCE_UNIT_COST (Resource, Resource_Unit_Cost) UNIT_USE (Product, Resource, Unit_Use) RESOURCE_AVAILABLE (Resource, Resource_Available) PROD_UNITS (Product, Prod_Units) Total_Profit = + of (Profit) Profit = (Unit_Profit) * (Prod_Units) Unit_Profit = (Unit_Price) - (Unit_Prod_Cost) Unit_Prod_Cost = + of (Uuse_Cost) by (Product) Uuse_Cost = (Resource_Unit_Cost) * (Unit_Use) Resource_Use = + of (Use) by (Resource) Use = (Unit_Use) * (Prod_Units) Enough = (Resource_Use ≤ Resource_Available) AND (Prod_Units ≥ 0) II. The EDB: omitted.

Features of the Functional MMS

The first desirable feature of an MMS is being knowledge-based. The basis on first-order logic implies that the functional MMS, at the macro level, is a knowledge-based system. The example of finding predecessors of "EOQ" DSM (Figure 7) is a simple application of a knowledge-based system. Moreover, under the universal relation assumption, DSMs can be selected and integrated based on descriptions in DSM predicates and their associated functional data bases. In other words, new DSMs can be formulated from choosing, integrating

and/or modifying descriptions of DSMs in a functional model base.

The second desirable feature of MMSs is flexibility. As illustrated above, the definitional system expressed in a functional data base is actually the one envisioned at the core of a structured model (Geoffrion 1987). Therefore, the functional DSM like structured modeling should be also widely applicable in the field of MS/OR (Geoffrion 1987). Furthermore, the functional MMS allows DSMs to be combined with data from a user and/or a data base since an EDB is itself a relational data base. It also permits data to be passed from another DSM. This can be done through use of DSM predicates in defining virtual attributes.

The third desirable feature of MMSs is independence. The functional MMS preserves representational independence between the mathematical structure and detailed data of a DSM. It represents the mathematical structure of a DSM as a set of definitions which is independent of the detailed data, a collection of relational tables. In other words, the mathematical structure of a DSM can be combined with different EDBs to generate different instances of a class of DSM. For example, the functional description of transportation DSM in Table 9 can be combined with another EDB shown in Figure 16a to formulate another DSM instance (Figure 16b). Moreover, the techniques used within the

functional MMS are independent of application domains; the functional approach can be used to describe DSMs of different application domains.

UNIT_TRANS_COST

(FactoryLoc,	CustomerLoc,	Unit_Trans_Cost)
Los Angeles	Denver	6.40
Los Angeles	Miami	17.20
Detroit	Denver	8.00
Detroit	Miami	8.64
New Orleans	Denver	8.12
New Orleans	Miami	5.44

SUPPLY (FactoryLoc, Supply)

Los Angeles	1,000
Detroit	1,500
New Orleans	1,200

DEMAND (CustomerLoc, Demand)

Denver	2,300
Miami	1,400

QTY (FactoryLoc, CustomerLoc, Qty)

			Corresponding Variable Names
Los Angeles	Denver	-	Qty_{11}
Los Angeles	Miami	-	Qty_{12}
Detroit	Denver	-	Qty_{21}
Detroit	Miami	-	Qty_{22}
New Orleans	Denver	-	Qty_{31}
New Orleans	Miami	-	Qty_{32}

Figure 16a. Another EDB of the Transportation DSM

$$\begin{aligned}
\text{Minimize} \quad & 6.40\ QTY_{11} + 17.20\ QTY_{12} + \\
& 8.00\ QTY_{21} + 8.64\ QTY_{22} + \\
& 8.12\ QTY_{31} + 5.44\ QTY_{32}
\end{aligned}$$

$$\begin{aligned}
\text{Subject to} \quad & QTY_{11} + QTY_{12} \leq 1{,}000 \\
& QTY_{21} + QTY_{22} \leq 1{,}500 \\
& QTY_{31} + QTY_{32} \leq 1{,}200 \\
& QTY_{11} + QTY_{21} + QTY_{31} = 2{,}300 \\
& QTY_{12} + QTY_{22} + QTY_{32} = 1{,}400 \\
& QTY_{11},\ QTY_{12},\ QTY_{21},\ QTY_{22},\ QTY_{31},\ QTY_{32} \geq 0
\end{aligned}$$

Figure 16b. The LP Formulation for the Transportation DSM in Figure 16a.

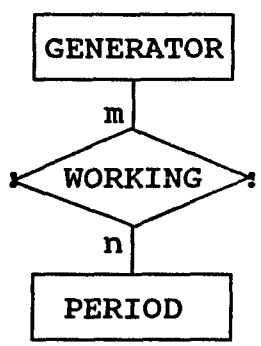
Finally, the fourth desirable feature of an MMS is to reflect a user's view of a DSM. Since mathematical associations of a DSM represent some down-to-earth connections between entities in the real world, users can better capture the semantic structure of a DSM if they perceive relationships between entities of a DSM. The functional MMS uses the E-R model, one of the best-known semantic model in data base design, to help users perceiving relationships between entities of a DSM. Additionally, the definitional system in an IDB, as described earlier, provides a natural way of developing a DSM in a hierarchical manner. Users can undertake the complexity of a DSM through defining virtual attributes.

Functional MMS versus Modeling Languages

In practice, there are several MMSs developed specifically for supporting the use of MP DSMs (e.g., PLATOFORM). Most of these systems provide users with special-purpose high-level MLs to formulate and enter an

MP DSM into a computer. Regarding the maintenance of values and variables referenced in an MP DSM, most of the MMSs use the generalized-array model as the major data organization. Though the generalized-array model, like a relational data base, views data as tables, the command syntax of the generalized-array model is quite different from that of a relational data base. In addition, the semantics of a class of DSM, represented in an ML, is still hidden in mathematical expressions with indices.

The similarities and differences between a functional MMS and a matrix generator using an ML can be demonstrated using a tariff rates problem drawn from Williams (1985b). The tariff rates problem is an integer programming (IP) DSM. The complete example is presented in Appendix A. The ERD and functional description of the tariff rates problem are shown in Figure 17 and Table 17 respectively.



GENERATOR (Generator, Gu_Available, Min_Level, Max_Level, Costph, Excostph, Scostpu)

WORKING (Period, Generator, Number, Nst, Out)

PERIOD (Period, Hours, Load_Demand)

Figure 17. The ERD for the Tariff Rates DSM

Table 17.--The Complete Functional Description of the Tariff Rates DSM

The Macro-level Description: A DSM Predicate

```
DSM (Tariff, [Upload, Hours, Load_Demand, Gu_Available,
      Min_Level, Max_Level, Costph, Excostph, Scostpu],
      [Number, Nst, Out], [min Total_Op_Cost], All)
```

The Micro-level Representation: a Functional Data Base

```
I. The IDB:
      UPLOAD (Upload)
      HOURS (Period, Hours)
      LOAD_DEMAND (Period, Load_Demand)
      GU_AVAILABLE (Generator, Gu_Available)
      MIN_LEVEL (Generator, Min_Level)
      MAX_LEVEL (Generator, Max_Level)
      COSTPH (Generator, Costph)
      EXCOSTPH (Generator, Excostph)
      SCOSTPU (Generator, Scostpu)
      NUMBER (Period, Generator, Number)
      NST (Period, Generator, Nst)
      OUT (Period, Generator, Out)
      Total_Op_Cost  = Total_Min_Cost + Total_Ex +
                       Total_Start_Cost
      Total_Min_Cost = + of (Min_Cost)
      Min_Cost       = Costph * Hours * Number
      Total_Ex       = + of (Ex_cost)
      Ex_Cost        = Excostph * Hours * (Out - Min_Out)
      Min_Out        = (Min_Level) * (Number)
      Total_Start_Cost    = + of (Start_Cost)
      Start_Cost     = Scostpu * Nst
      Period_Out     = + of (Out) by (Period)
      Period_Max_Out = + of (Max_Out) by (Period)
      Max_Out        = (Max_Level) * (Number)
      Extra_Demand   = (Upload) * (Load_Demand)
      Num_Increased  = (Number) - (Pre_Number)
      Pre_Number     = (pre) of (Number) order (Period) by
                       (Generator)
      All = (Period_Out ≥ Load_Demand) AND
          (Period_Max_Out ≥ Extra_Demand) AND
          (Out ≥ Min_Out) AND (Out ≤ Max_Out) AND
          (Nst ≥ Num_Increased) AND
          (Number ≤ Gu_Available) AND (Nst ≤ Gu_Available)
          AND (Number ≥ 0) AND (Nst ≥ 0) AND (Out ≥ 0)
```

Table 17.--The Complete Functional Description of the Tariff Rates DSM (Continued)

The Micro-level Representation: A Functional Data Base

II. The EDB:

HOURS

(Period,	Hours)
1	6
2	3
3	6
4	3
5	6

LOAD_DEMAND

(Period,	Load_Demand)
1	15000
2	30000
3	25000
4	40000
5	27000

GU_AVAILABLE

(Generator,	Gu_Available)
G1	12
G2	10
G3	5

COSTPH

(Generator,	Costph)
G1	1000
G2	2600
G3	3000

MIN_LEVEL

(Generator,	Min_Level)
G1	850
G2	1250
G3	1500

MAX_LEVEL

(Generator,	Max_Level)
G1	2000
G2	1750
G3	4000

EXCOSTPH

(Generator,	Excostph)
G1	2
G2	1.3
G3	3

SCOSTPU

(Generator,	Scostpu)
G1	2000
G2	1000
G3	500

NUMBER

(Period,	Generator,	Number)	Corresponding Decision Variables
1	G1	-	X_{11}
2	G1	-	X_{21}
3	G1	-	X_{31}
4	G1	-	X_{41}
5	G1	-	X_{51}
1	G2	-	X_{12}
2	G2	-	X_{22}
3	G2	-	X_{32}
4	G2	-	X_{42}
5	G2	-	X_{52}
1	G3	-	X_{13}
2	G3	-	X_{23}
3	G3	-	X_{33}
4	G3	-	X_{43}
5	G3	-	X_{53}

Table 17.--The Complete Functional Description of the Tariff Rates DSM (Continued)

II. The EDB (Continued):

NST (Period, Generator, Nst)			Corresponding Decision Variables
1	G1	-	Y_{11}
2	G1	-	Y_{21}
3	G1	-	Y_{31}
4	G1	-	Y_{41}
5	G1	-	Y_{51}
1	G2	-	Y_{12}
2	G2	-	Y_{22}
3	G2	-	Y_{32}
4	G2	-	Y_{42}
5	G2	-	Y_{52}
1	G3	-	Y_{13}
2	G3	-	Y_{23}
3	G3	-	Y_{33}
4	G3	-	Y_{43}
5	G3	-	Y_{53}

OUT (Period, Generator, Out)			Corresponding Decision Variables
1	G1	-	Z_{11}
2	G1	-	Z_{21}
3	G1	-	Z_{31}
4	G1	-	Z_{41}
5	G1	-	Z_{51}
1	G2	-	Z_{12}
2	G2	-	Z_{22}
3	G2	-	Z_{32}
4	G2	-	Z_{42}
5	G2	-	Z_{52}
1	G3	-	Z_{13}
2	G3	-	Z_{23}
3	G3	-	Z_{33}
4	G3	-	Z_{43}
5	G3	-	Z_{53}

Figure 18 (Williams 1985a) presents the tariff rates DSM in MAGIC, an ML. According to Williams (1985a), the advantages of MAGIC (Williams 1985a) include mirroring conventional mathematical notation, automatic indexing and repetition, allowing relationships between indices,

separating data from structure of a DSM, natural input format, easier debugging and modification, etc.

```
NAME: TARIFF;                              (Job name)

MAXI=5;                                    (Subscript assignments)
MAXJ=3;

UPLOAD = 1.15                              (Parameter assignments)
HOURS (MAXI) = 6, 3, 6, 3, 6;
LOAD_DEMAND (MAXI) = 15000, 30000, 25000, 40000, 27000;
GU_AVAILABLE (MAXJ) = 12, 10, 5;
MIN_LEVEL (MAXJ) = 850, 1250, 1500;
MAX_LEVEL (MAXJ) = 2000, 1750, 4000;
COSTPH    (MAXJ) = 1000, 2600, 3000;
EXCOSTPH  (MAXJ) = 2, 1.3, 3;
SCOSTPU   (MAXJ) = 2000, 1000, 500;

NUMBER (MAXI, MAXJ) INTEGER;               (Variable definitions)
NST (MAXI, MAXJ) INTEGER;
OUT (MAXI, MAXJ);

                                           (Objective function)
TOTAL_OP_COST (MIN): SIGMA I=1, MAXI: SIGMA J=1, MAXJ:
{EXCOSTPH(J)*HOURS(I)* [OUT(I,J)-MIN_LEVEL(J)*NUMBER(I,J)]
+ COSTPH(J) *HOURS(I)* NUMBER(I,J) + SCOSTPU(J)* NST(I,J)}

                                           (Constraint statements)
PERIOD_OUT (I=1, MAXI):
   SIGMA J=1,MAXJ: OUT (I,J) >= LOAD_DEMAND (I);
PERIOD_MAX_OUT (I=1, MAXI): SIGMA J=1, MAXJ:
    (MAX_LEVEL(J) * NUMBER(I, J) >= UPLOAD*LOAD_DEMAND(I);

MIN (I=1,MAXI:J=1,MAXJ):
   OUT (I,J) - MIN_LEVEL (J)*NUMBER (I,J) >= 0;
MAX (I=1,MAXI:J=1,MAXJ):
   OUT (I,J) - MAX_LEVEL (J)*NUMBER (I,J) <= 0;
ST (I=1,MAXI:J=1,MAXJ, I>1):
   NST (I,J) - NUMBER (I,J) + NUMBER (I-1,J) >=0;
ST (I=1,MAXI:J=1,MAXJ, I=1):
   NST (I,J) - NUMBER (I,J) + NUMBER (MAXI,J) >=0;
BOUNDS(I=1,MAXI:J=1,MAXJ): NUMBER(I,J) <= GU_AVAILABLE(J);
BOUNDS(I=1,MAXI:J=1,MAXJ): NST (I,J)   <= GU_AVAILABLE(J);

MPS TARIFILE;        (MPSX output format to file TARIFILE)
STOP;
```

Figure 18. The Tariff Rates DSM in MAGIC (Williams 1985a)

By comparing Table 17 and Figure 18, several similarities can be observed. First, the EDB in Table 17 is parallel to sections of parameter assignments and variable definitions in Figure 18. Second, the first six definitions of virtual numeric attributes in the IDB are corresponding to the section of objective function of the MAGIC statements. Finally, each condition defined in the logical attribute "All" are equivalent to the corresponding MAGIC constraint statement. In fact, the functional DSM preserves the features of the ML, MAGIC. It mirrors conventional mathematical notation, permits relationships between indices through the operator "pre", separates the data from the structure of a DSM, adopts relational data base, and allows easy modification.

However, the functional DSM is superior to the one in MAGIC. First of all, the IDB does not use indices; MAGIC does. Without indices, the mathematical structure of the DSM is presented in a clearer way and is easier to read and understand. Even so, a functional DSM still allows automatic indexing and repetition by using implicit tuple variables of which the range are tuples of relational tables. Secondly, the IDB captures the semantics of the DSM by using meaningful virtual attribute names; while in MAGIC statements, the semantics are hidden in the mathematical expressions.

For example, it is obvious that, from the definition

in the IDB, "Total_Op_Cost" is the sum of the total basic generator operation cost "Total_Min_Cost", the total extra generator operation cost "Total_Ex", and the total generator start-up cost "Total_Start_Cost". Nevertheless, the components of "Total_Op_Cost" is not as obvious in the DSM using MAGIC.

CHAPTER 5

USES OF THE FUNCTIONAL MMS

The functional MMS can be of several uses: top-down modeling, and DSM combination and/or integration. Top-down modeling is a concept of getting the big picture of a DSM right at the outset with minimum distraction and developing details of the DSM in stages. The functional MMS provides ways of developing DSMs in a hierarchical manner to deal with the complexity of a DSM. Moreover, the functional MMS allows DSM builders to construct a total DSM by integrating DSMs selected from a functional model base.

Top-Down Modeling

The functional MMS, as discussed earlier, provides a natural way of undertaking the complexity of a DSM through defining virtual attributes. Defining a virtual attribute in terms of other virtual attributes repeatedly represents a series of stepwise refinements based on a hierarchical view of a decision problem.

Users can also develop a complex DSM by embedding DSM predicates of predecessors in the IDB of a total DSM. Use of embedded DSM predicates in top-down modeling can be

demonstrated using the classic economic order quantity (EOQ) DSM with multiple independent items. The example is drawn from Geoffrion (1987).

The Classic EOQ DSM

The classic EOQ DSM is to determine the economic order quantity to minimize the total cost including setup and carrying costs. Parameter attributes of the classic EOQ DSM include the demand rate (units per year), holding cost rate (dollars per unit per year), and fixed setup cost (dollars per setup) of each item. Table 18 depicts the functional description of the classic EOQ DSM.

Table 18.--The Functional Description of the Classic EOQ DSM

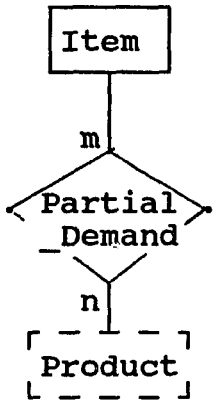
The Macro-level Description: A DSM Predicate
DSM (EOQ, [Demand_Rate, Hold_Cost, Fixed_Cost], [Order_Qty], [(min Total_Order_Cost)], -)
The Micro-level Representation: A Functional Data Base
I. The IDB: DEMAND_RATE (Item, Demand_Rate) HOLD_COST (Item, Hold_Cost) FIXED_COST (Item, Fixed_Cost) ORDER_QTY (Item, Order_Qty) Total_Order_Cost= + of (Item_Cost) Item_Cost = (Setup_Cost) + (Carrying_Cost) Setup_Cost = (Setup_Frequency) * (Fixed_Cost) Setup_Frequency = (Demand_Rate) / (Order_Qty) Carrying_Cost = (Hold_Cost) * (Order_Qty) / 2
II. An EDB: omitted.

The EOQ DSM (1)

In a real application, the demand rate, holding cost, and fixed cost of an item must be calculated from other data. Assume the following.

(1) The demand rate of an item must be derived from demands of final products.
(2) The holding cost rate is the sum of the opportunity cost of capital tied up and the out-of-pocket storage cost.
(3) The fixed setup cost is the sum of separate costs for the materials and labor consumed.

The complete EOQ DSM "EOQ1" can be constructed by calling predecessors for values of the parameter attributes. The ERD and functional description of the DSM "EOQ1" are shown in Figure 19 and Table 19, respectively. Three predecessors, "Demand1", "Hold_Cost1", and "Fixed_Cost1" are called in the IDB to provide data for parameter attributes, and then the classic EOQ DSM is called to complete the description. The parameter attributes of "EOQ1" are the union of the parameter attributes of all the predecessors unless they are provided by the solution of some other predecessor.



ITEM (Item, Unit_Value, Storage_Cost,
Fixed_Material_Cost,
Fixed_Labor_Cost, Order_Qty)

PARTIAL_DEMAND
(Item, Product, Partial_Demand)

Figure 19. The ERD of the DSM "EOQ1"

Table 19.--The Functional Description of the DSM "EOQ1"

The Macro-level Description: A DSM Predicate
DSM (EOQ1, [Partial_Demand, OC_Rate, Unit_Value, Storage_Cost, Fixed_Material_Cost, Fixed_Labor_Cost], [Order_Qty], [(min Total_Order_Cost)], -)
The Micro-level Representation: A Functional Data Base
I. The IDB: PARTIAL_DEMAND (Item, Product, Partial_Demand) OC_RATE (OC_Rate) UNIT_VALUE (Item, Unit_Value) STORAGE_COST (Item, Storage_Cost) FIXED_MATERIAL_COST (Item, Fixed_Material_Cost) FIXED_LABOR_COST (Item, Fixed_Labor_Cost) ORDER_QTY (Item, Order_Qty) DSM (Demand1, [Partial_Demand], [Demand_Rate], [], -) DSM (Hold_Cost1, [OC_Rate, Unit_Value, Storage_Cost], [Hold_Cost], [], -) DSM (Fixed_Cost1, [Fixed_Material_Cost, Fixed_Labor_Cost], [Fixed_Cost], [], -) DSM (EOQ, [Demand_Rate, Hold_Cost, Fixed_Cost], [Order_Qty], [(min Total_Order_Cost)], -) II. The EDB: omitted.

Under the first assumption, the demand rate of an item is calculated as the sum of partial demand rates of the final products. Each final product contributes a partial demand rate (units per year) for each item. The ERD and functional description of the DSM "Demand1" are shown in Figure 20 and Table 20 respectively.

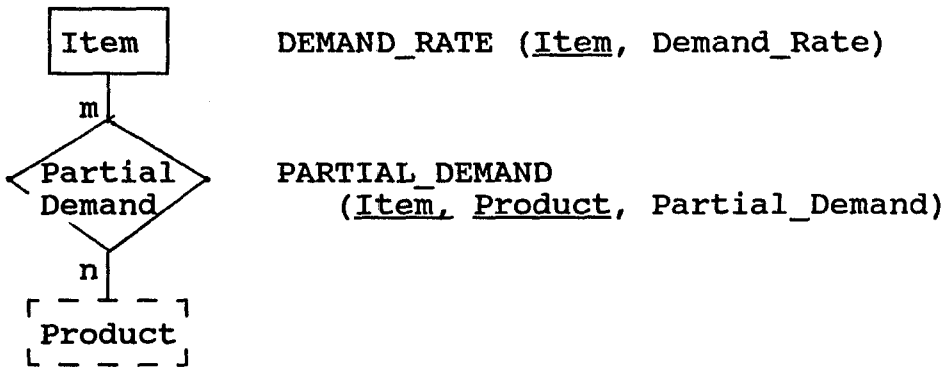


Figure 20. The ERD for the DSM "Demand1"

Table 20.--The Functional Description of the DSM "Demand1"

The Macro-level Description: A DSM Predicate
DSM (Demand1, [Partial_Demand], [Demand_Rate], [], -)
The Micro-level Representation: A Functional Data Base
I. The IDB: PARTIAL_DEMAND (Item, Product, Partial_Demand) Demand_Rate = + of (Partial_Demand) by (Item) II. The EDB: omitted.

Under the second assumption, the holding cost rate of an item is the sum of the opportunity cost of capital tied up and the out-of-pocket storage cost. Table 21

shows the functional description of the DSM "Hold_Cost1".

Table 21.--The Functional Description of the DSM "Hold_Cost1"

The Macro-level Description: A DSM Predicate
DSM (Hold_Cost1, [OC_Rate, Unit_Value, Storage_Cost], [Hold_Cost], [], -)
The Micro-level Representation: A Functional Data Base
I. The IDB: OC_RATE (OC_Rate) UNIT_VALUE (Item, Unit_Value) STORAGE_COST (Item, Storage_Cost) Hold_Cost = (Opportunity_Cost) + (Storage_Cost) Opportunity_Cost = (OC_Rate) * (Unit_Value)
II. The EDB: omitted.

Under the third assumption, the fixed setup cost of an item is the sum of separate costs for the materials and labor consumed. The functional description of the DSM "Fixed_Cost1" is shown in Table 22.

The EOQ DSM (2)

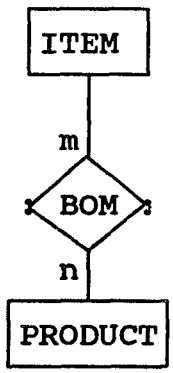
Greater detail can be added to the DSM "EOQ1". Assume the following.

(1) The partial demand rates of final products for each item are computed from estimated demands for final products and the parts explosion.
(2) The unit value of each item must be assembled from its major components.
(3) The setup labor cost must be calculated as labor hours times hour rate.

Table 22.--The Functional Description of the DSM "Fixed_Cost1"

The Macro-level Description: A DSM Predicate
DSM (Fixed_Cost1, [Fixed_Material_Cost, Fixed_Labor_Cost], [Fixed_Cost], [], -)
The Micro-level Representation: A Functional Data Base
I. The IDB: FIXED_MATERIAL_COST (Item, Fixed_Material_Cost) FIXED_LABOR_COST (Item, Fixed_Labor_Cost) Fixed_Cost = Fixed_Material_Cost + Fixed_Labor_Cost
II. The EDB: omitted.

The more complicated EOQ DSM ("EOQ2") can be constructed by calling predecessors with more complex calculations. The ERD and functional description of the DSM "EOQ2" are shown in Figure 21 and Table 23, respectively. Similar to "EOQ1", the DSM "EOQ2" includes DSM predicates of three predecessors "Demand2", "Hold_Cost2", and "Fixed_Cost2" to provide input data and the DSM predicate of the classic EOQ DSM.



ITEM (Item, Storage_Cost, Direct_Material, Direct_Labor, Setup_Hours, Fixed_Material_Cost, Order_Qty)

BOM (Item, Product, Bom)

PRODUCT_DEMAND (Product, Product_Demand)

Figure 21. The ERD of the DSM "EOQ2"

Table 23.--The Functional Description of the DSM "EOQ2"

The Macro-level Description: A DSM Predicate

```
DSM (EOQ2, [Bom, Product_Demand, OC_Rate, Storage_Cost,
      Direct_Material, Direct_Labor, Setup_Rate,
      Setup_Hours, Fixed_Material_Cost], [Order_Qty],
      [(min Total_Order_Cost)], -)
```

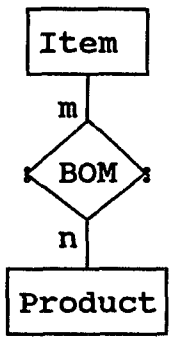
The Micro-level Representation: A Functional Data Base

I. The IDB:

```
      BOM (Item, Product, Bom)
      PRODUCT_DEMAND (Product, Product_Demand)
      OC_RATE          (OC_Rate)
      STORAGE_COST     (Item, Storage_Cost)
      DIRECT_MATERIAL (Item, Direct_Material)
      DIRECT_LABOR     (Item, Direct_Labor)
      SETUP_RATE  (Setup_Rate)
      SETUP_HOURS (Item, Setup_Hours)
      FIXED_MATERIAL_COST (Item, Fixed_Material_Cost)
      ORDER_Qty   (Item, Order_Qty)
      DSM (Demand2, [Product_Demand, Bom], [Demand_Rate],
          [], -)
      DSM (Hold_Cost2, [OC_Rate, Direct_Material,
          Direct_Labor, Storage_Cost], [Hold_Cost], [], -)
      DSM (Fixed_Cost2, [Setup_Rate, Setup_Hours,
          Fixed_Material_Cost], [Fixed_Cost], [], -)
      DSM (EOQ, [Demand_Rate, Hold_Cost, Fixed_Cost],
          [Order_Qty], [(min Total_Order_Cost)], -)
```

II. The EDB: omitted.

Under the first assumption, the demand rate of each item is calculated as the sum of demands derived from final products. Furthermore, the partial demand rate of a final product for an item is built up from demand estimates for final products and the parts explosion. The ERD and functional description of the DSM "Demand2" are shown in Figure 22 and Table 24, respectively.



DEMAND_RATE (Item, Demand_Rate)

BOM (Item, Product, Bom)

PRODUCT_DEMAND (Product, Product_Demand)

Figure 22. The ERD of the DSM "Demand2"

Table 24.--The Functional Description of the DSM "Demand2"

The Macro-level Description: A DSM Predicate
DSM (Demand2, [Bom, Product_Demand], [Demand_Rate], [], -)
The Micro-level Representation: A Functional Data Base
I. The IDB: BOM (Item, Product, Bom) PRODUCT_DEMAND (Product, Product_Demand) Partial_Demand = (Bom) * (Product_Demand) Demand_Rate = + of (Partial_Demand) by (Item)
II. The EDB: omitted.

According to the second assumption, the holding cost rate of each item is the sum of the opportunity cost of capital tied up and the out-of-pocket storage cost. The unit value of an item is assembled from its major components, direct material and labor costs. The functional description of the DSM "Hold_Cost2" is shown in Table 25.

Table 25.--The Complete Functional Description of the DSM "Hold_Cost2"

The Macro-level Description: A DSM Predicate
DSM (Hold_Cost2, [OC_Rate, Direct_Material, Direct_Labor, Storage_Cost], [Hold_Cost], [], -)
The Micro-level Representation: A Functional Data Base
I. The IDB: OC_RATE (OC_Rate) STORAGE_COST (Item, Storage_Cost) DIRECT_MATERIAL (Item, Direct_Material) DIRECT_LABOR (Item, Direct_Labor) Unit_Value = (Direct_Material) + (Direct_Labor) Opportunity_Cost = (OC_Rate) * (Unit_Value) Hold_Cost = (Opportunity_Cost) + (Storage_Cost)
II. The EDB: omitted.

The fixed cost of each item, based on the third assumption, is the sum of separate costs for the materials and labor consumed. In addition, the setup labor cost is the product of labor units times labor rate. The functional description of the DSM "Fixed_Cost2" is shown in Table 26.

DSM Combination and Integration

In the real life, most very large DSMs arise through integrating smaller DSMs. DSM integration means coordinated unification of two or more distinct DSM instances or classes. The functional MMS provides a convenient framework for DSM integration because it makes explicit what must be coordinated, namely definitional and computational dependencies among numeric attributes. The

Table 26.--The Complete Functional Description of the DSM "Fixed_Cost2"

The Macro-level Description: A DSM Predicate
DSM (Fixed_Cost2, [Setup_Rate, Setup_Hours, Fixed_Material_Cost], [Fixed_Cost], [], -)
The Micro-level Representation: A Functional Data Base
I. The IDB: SETUP_RATE (Setup_Rate) SETUP_HOURS (Item, Setup_Hours) FIXED_MATERIAL_COST (Item, Fixed_Material_Cost) Fixed_Labor_Cost = Setup_Rate * Setup_Hours Fixed_Cost = Fixed_Material_Cost + Fixed_Labor_Cost II. The EDB: omitted.

functional MMS allows DSMs to be integrated at the DSM instance or DSM schema level. DSM instances of the same class can be integrated to construct a more complex DSM; functional DSMs of different classes can also be integrated to form a total DSM.

Integrating Two Instances of a DSM

When DSM instances of the same class are to be integrated, it usually indicates that more entity types are involved. As a result, some primitive relations incorporate more symbolic attributes in their identifiers to accommodate the participation of additional entity types. The virtual relations defined on these primitive attributes may also include more symbolic attributes in their identifiers. Due to the increasing dimension of some primitive relations, the dimension of the integrated

DSM also increases.

The integration of two instances of the same DSM class can be illustrated by the example of a multi-plant product mix DSM (Williams 1985b). The example is to show how to construct a nonseparable total DSM by integrating different DSM instances of the product mix DSM.

Assume a company has two plants, A and B. Each plant manufactures two products, "standard" and "deluxe". The unit profit of "standard" is $10, while "deluxe" is $15. Two processes, grinding and polishing, are used to produce the products. The grinding and polishing times (in hours) available in each plant and needed for producing one unit of each product in each plant are given in Table 27. In addition, each unit of each product uses 4 kilograms of a raw material ("raw"). The company has 120 kilograms of "raw" available per week.

Table 27.--The Information Regarding Capacities of Grinding and Polishing

	Grinding	Polishing
Plant A	80	60
"standard"	4	2
"deluxe"	2	5
Plant B	60	75
"standard"	5	5
"deluxe"	3	6

If the company arbitrarily allocates 75 kilograms of "raw" to plant A per week and 45 kilograms to plant B, the multi-plant product mix DSM is the combination of two

single-plants. Each is an instance of the single-plant product mix DSM of which the ERD is presented in Figure 23; the functional description is presented Table 28. The EDBs for plants A and B are depicted in Figures 24 and 25, respectively.

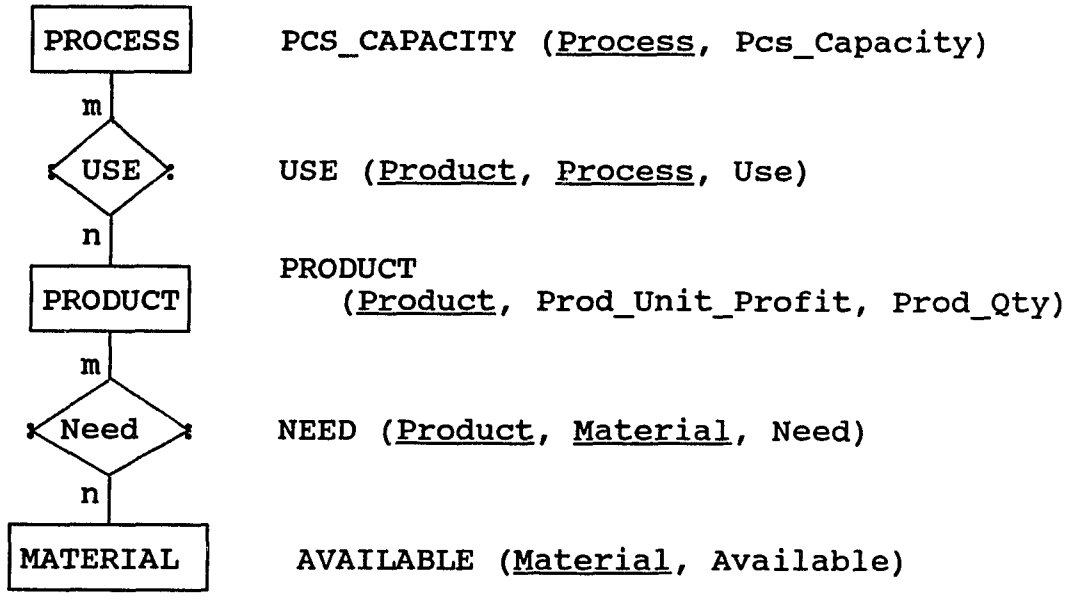


Figure 23. The ERD for the Single-Plant Product Mix DSM

Table 28.--The Complete Functional Description of the Single-Plant Product Mix DSM

The Macro-level Description: A DSM Predicate

```
DSM (Single_Plant, [Prod_Unit_Profit, Use, Pcs_Capacity,
      Need, Available], [Prod_Qty], [(max Total_Profit)],
      Enough)
```

The Micro-level Representation: A Functional Data Base

I. The IDB:

```
PROD_UNIT_PROFIT (Product, Prod_Unit_Profit)
USE (Product, Process, Use)
PCS_CAPACITY (Process, Pcs_Capacity)
NEED (Product, Material, Need)
AVAILABLE (Material, Available)
PROD_QTY (Product, Prod_Qty)
Total_Profit   = + of (Profit)
Profit         = (Prod_Unit_Profit) * (Prod_Qty)
Process_Use    = + of (Product_Use) by (Process)
Product_Use    = (Use) * (Prod_Qty)
Material_Need = + of (Product_Need) by (Material)
Product_Need   = (Need) * (Prod_Qty)
Enough = (Process_Use ≤ Pcs_Capacity) AND
         (Materials_Need ≤ Available) AND
         (Prod_Qty ≥ 0)
```

II. The EDBs: see Figures 24 and 25.

PROD_UNIT_PROFIT

(Product,	Prod_Unit_Profit)
standard	10
deluxe	15

NEED

(Product,	Material,	Need)
standard	raw	4
deluxe	raw	4

USE

(Product,	Process,	Use)
standard	Grinding	4
standard	Polishing	2
deluxe	Grinding	2
deluxe	Polishing	5

PCS_CAPACITY

(Process,	Pcs_Capacity)
Grinding	80
Polishing	60

AVAILABLE (Material, Available)

raw	75

PROD_QTY	(Product,	Prod_Qty)	Corresponding Variable Names
	standard	-	Qty_1
	deluxe	-	Qty_2

Figure 24. The EDB for Plant A

PROD_UNIT_PROFIT

(Product,	Prod_Unit_Profit)
standard	10
deluxe	15

NEED

(Product,	Material,	Need)
standard	raw	4
deluxe	raw	4

USE

(Product,	Process,	Use)
standard	Grinding	5
standard	Polishing	5
deluxe	Grinding	3
deluxe	Polishing	6

PCS_CAPACITY

(Process,	Pcs_Capacity)
Grinding	60
Polishing	75

AVAILABLE (Material, Available)

raw	45

PROD_QTY	(Product,	Prod_Qty)	Corresponding Variable Names
	standard	-	Qty_3
	deluxe	-	Qty_4

Figure 25. The EDB for Plant B

The multi-plant product mix DSM can now be constructed by integrating the two DSM instances in Figures 24 and 25. Because relations PROD_UNIT_PROFIT and NEED are independent of plants, their data rows can be combined to generate the data of the integrated relations. However, the integration of relations USE, PCS_CAPACITY and PROD_QTY is not as straightforward, since they are dependent on plants. Due to the participation of entity type PLANT, these three relations need to include symbolic attribute "Plant" in their identifiers. Finally, the integration of relation AVAILABLE requires extra attention because both plants compete the use of the material. It is up to a DSM builder to decide how to integrate the corresponding tuples of two DSM instances. In this case, the integrated relation AVAILABLE should contain only one tuple which represents the raw material constraint of the company, not of either plant.

The ERD of the multi-plant product mix DSM is presented in Figure 26. It is obvious that the major difference between this ERD and the one for single-plant DSM is that entity type PLANT takes part in several relationship types. The complete functional description for the multi-plant product mix DSM is presented in Table 29.

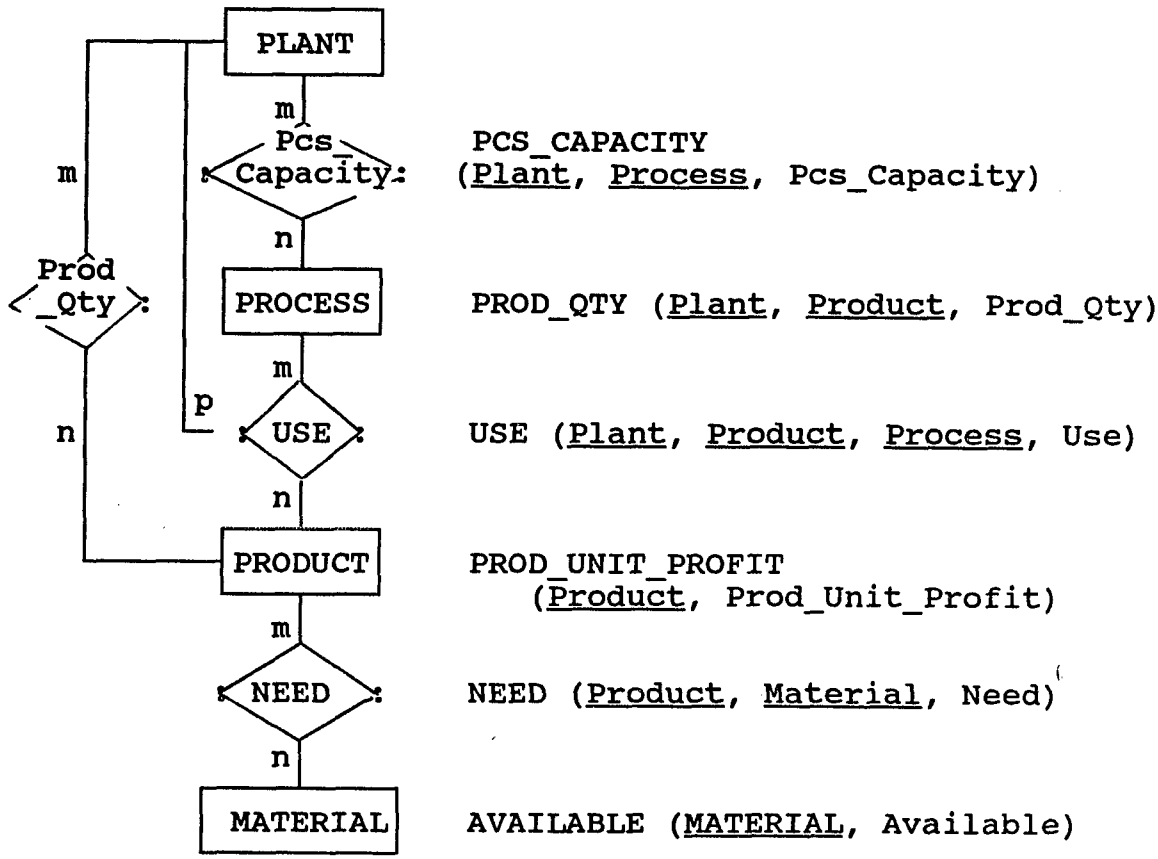


Figure 26. The ERD for the Multi-Plant Product Mix DSM

Table 29.--The Complete Functional Description of the Multi-Plant Product Mix DSM

The Macro-level Description: A DSM Predicate

```
DSM (Multi_Plant, [Prod_Unit_Profit, Use, Pcs_Capacity,
      Need, Available], [Prod_Qty], [(max Total_Profit)],
      Enough)
```

The Micro-level Representation: A Functional Data Base

I. The IDB:

```
PROD_UNIT_PROFIT (Product, Prod_Unit_Profit)
USE (Plant, Product, Process, USE)
PCS_CAPACITY (Plant, Process, Pcs_Capacity)
NEED (Product, Material, Need)
AVAILABLE (Material, Available)
PROD_QTY (Plant, Product, Prod_Qty)
Total_Profit   = + of (Profit)
Profit         = (Prod_Unit_Profit) * (Prod_Qty)
Process_Use    = + of (Product_Use)
                   by (Process, Plant)
Product_Use    = (Use) * (Prod_Qty)
Materials_Need = + of (Product_Need) by (Material)
Product_Need   = (Need) * (Prod_Qty)
Enough = (Process_Use ≤ Pcs_Capacity) AND
         (Materials_Need ≤ Available) AND
         (Prod_Qty ≥ 0)
```

II. The EDB:

PROD_UNIT_PROFIT

(Product,	Prod_Unit_Profit)
standard	10
deluxe	15

NEED

(Product,	Material,	Need)
standard	raw	4
deluxe	raw	4

USE

(Plant,	Product,	Process,	Use)
A	standard	Grinding	4
B	standard	Grinding	5
A	standard	Polishing	2
B	standard	Polishing	5
A	deluxe	Grinding	2
B	deluxe	Grinding	3
A	deluxe	Polishing	5
B	deluxe	Polishing	6

PCS_CAPACITY

(Plant,	Process,	Pcs_ Capacity)
A	Grinding	80
B	Grinding	60
A	Polishing	60
B	Polishing	75

AVAILABLE (Material, Available)

raw	120

Table 29.--The Complete Functional Description of the Multi-Plant Product Mix DSM (Continued)

The Micro-level Representation: A Functional Data Base			
II. The EDB (Continued):			
PROD_QTY (Plant, Product, Prod_Qty)			Corresponding Variable Names
A	standard	-	Qty_1
A	deluxe	-	Qty_2
B	standard	-	Qty_3
B	deluxe	-	Qty_4

Combining and Integrating Two Distinct DSMs

When two or more distinct DSMs share common attributes, they can be used sequentially to form a total DSM. However, sequential use of DSMs usually leads to sub-optimization. To find the global optimization, it requires coordinated unification of common numeric attributes; DSMs need to be integrated to form the total DSM. The integration of DSM schema can be accomplished by applying the unification algorithm (Rich 1983), a matching procedure to discover a set of substitutions for binding attributes of two DSM predicates together. Since in an EDB the dimension of a numeric attribute is specified by the number of values taken by the identifier, the unification algorithm needs to match dimensions of attributes as well as to bind attributes.

The integration of two distinct DSMs is illustrated by the example adopted from Geoffrion (1987): integrating classic transportation and EOQ DSMs. The example is also used to demonstrate the distinction between DSM

integration and DSM combination. For the sake of convenience, the ERD and functional description of the standard transportation DSM are reproduced in Figure 27 and Table 30, respectively.

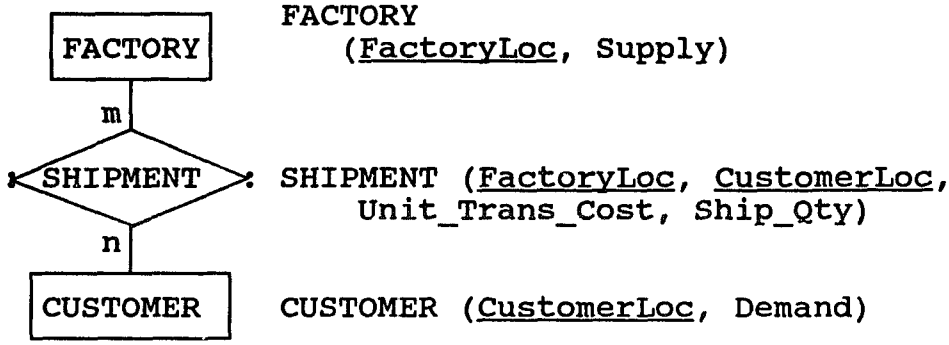


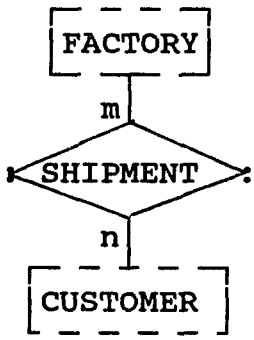
Figure 27. The ERD for the Transportation DSM

Table 30.--The Complete Functional Description of the Standard Transportation DSM

The Macro-level Description: A DSM Predicate
DSM (Transport, [Unit_Trans_Cost, Supply, Demand], [Ship_Qty], [(min Total_Trans_Cost)], Meet)
The Micro-level Representation: A Functional Data Base
I. The IDB: UNIT_TRANS_COST (FactoryLoc, CustomerLoc, Unit_Trans_Cost) SUPPLY (FactoryLoc, Supply) DEMAND (CustomerLoc, Demand) SHIP_QTY (FactoryLoc, CustomerLoc, Ship_Qty) Total_Trans_Cost = + of (Trans_Cost) Trans_Cost = (Unit_Trans_Cost) * (Ship_Qty) Qty_Supplied = + of (Ship_Qty) by (FactoryLoc) Qty_Received = + of (Ship_Qty) by (CustomerLoc) Meet = (Qty_Supplied ≤ Supply) AND (Qty_Received = Demand) AND (Ship_Qty ≥ 0) II. The EDB: omitted.

The classic EOQ DSM used to demonstrate DSM integration is with the following modifications (Figure 28 and Table 31).

(1) Each transportation link (FactoryLoc, CustomerLoc) plays the role of an "item" (Item) in the original EOQ problem.
(2) Each transportation flow (Ship_Qty) plays the role of a "demand rate" (Demand_Rate).
(3) A setup cost (Setup_Cost) is reinterpreted as a shipment receiving cost (Receiving_Cost).



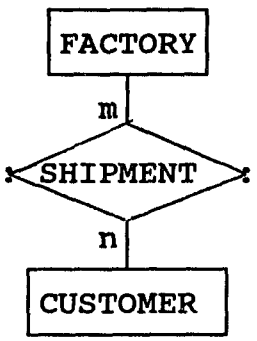
SHIPMENT
(FactoryLoc, CustomerLoc, Hold_Cost)
Fixed_Cost, Ship_Qty, Order_Qty)

Figure 28. The ERD for the Modified EOQ

Table 31. --The Complete Functional Description of the Modified EOQ DSM

The Macro-level Description: A DSM Predicate
DSM (EOQ, [Ship_Qty, Hold_Cost, Fixed_Cost], [Order_Qty], [(min Total_Ship_Cost)], -)
The Micro-level Representation: A Functional Data Base
I. The IDB: SHIP_QTY (FactoryLoc, CustomerLoc, Ship_Qty) HOLD_COST (FactoryLoc, CustomerLoc, Hold_Cost) FIXED_COST (FactoryLoc, CustomerLoc, Fixed_Cost) ORDER_QTY (FactoryLoc, CustomerLoc, Order_Qty) Total_Ship_Cost = + of (Shipment_Cost) Shipment_Cost = (Receiving_Cost) + (Carrying_Cost) Receiving_Cost = (Receiving_Freq) * (Fixed_Cost) Receiving_Freq = (Ship_Qty) / (Order_Qty) Carrying_Cost = (Hold_Cost) * (Order_Qty) / 2
II. The EDB: omitted.

Since the transportation and modified EOQ DSMs share the attribute "Ship_Qty", they can be used sequentially to construct a total DSM. The standard transportation DSM can be used first to solve for the attribute "Ship_Qty". Then attribute "Order_Qty" is chosen by the modified EOQ DSM for its closed form solution. DSM predicates of both the transportation and modified EOQ DSMs can be embedded in the combined DSM as submodels. The ERD and functional description of the combined DSM are shown in Figure 29 and Table 32, respectively.



FACTORY
(FactoryLoc, Supply)

SHIPMENT (FactoryLoc, CustomerLoc,
Unit_Trans_Cost, Hold_Cost, Fixed_Cost,
Ship_Qty, Order_Qty)

CUSTOMER (CustomerLoc, Demand)

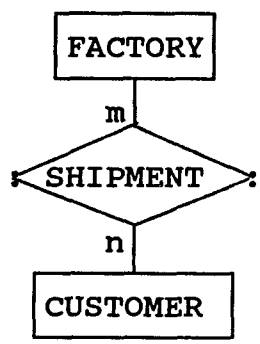
Figure 29. The ERD for the Combined DSM

Table 32.--The Complete Functional Description of the Combined DSM

The Macro-level Description: A DSM Predicate
DSM (Combine, [Unit_Trans_Cost, Supply, Demand, Hold_Cost, Fixed_Cost], [Ship_Qty, Order_Qty, Total_Cost], [(min Total_Trans_Cost), (min Total_Ship_Cost)], Meet)
The Micro-level Representation: A Functional Data Base
I. The IDB: UNIT_TRANS_COST (FactoryLoc, CustomerLoc, Unit_Trans_Cost) SUPPLY (FactoryLoc, Supply) DEMAND (CustomerLoc, Demand) HOLD_COST (FactoryLoc, CustomerLoc, Hold_Cost) FIXED_COST (FactoryLoc, CustomerLoc, Fixed_Cost) DSM (Transport, [Unit_Trans_Cost, Supply, Demand], [Ship_Qty], [(min Total_Trans_Cost)], Meet) DSM (EOQ, [Ship_Qty, Hold_Cost, Fixed_Cost], [Order_Qty], [(min Total_Ship_Cost)], -) Total_Cost = Total_Trans_Cost + Total_Ship_Cost II. The EDB: omitted.

However, sequential use of two DSMs leads to sub-optimization. To find jointly optimal choices of both decision variables "Ship_Qty" and "Order_Qty" the two DSMs must be integrated. This can be accomplished by

concatenating the mathematical structures of the two DSMs. Furthermore, instead of two separate objection functions, there should be only one objective function which requires the global optimization to be solved. The ERD and functional description of the integrated DSM are given in Figure 30 and Table 33.



FACTORY
(FactoryLoc, Supply)

SHIPMENT (FactoryLoc, CustomerLoc,
Unit_Trans_Cost, Hold_Cost, Fixed_Cost,
Ship_Qty, Order_Qty)

CUSTOMER (CustomerLoc, Demand)

Figure 30. The ERD for the Integrated DSM

Table 33.--The Complete Functional Description of the Integrated DSM

The Macro-level Description: A DSM Predicate

```
DSM (Integrate, [Unit_Trans_Cost, Supply, Demand,
      Hold_Cost, Fixed_Cost], [Ship_Qty, Order_Qty],
      [(min Total_Cost), Meet)
```

The Micro-level Representation: A Functional Data Base

```
I. The IDB:
      UNIT_TRANS_COST
          (FactoryLoc, CustomerLoc, Unit_Trans_Cost)
      SUPPLY (FactoryLoc, Supply)
      DEMAND (CustomerLoc, Demand)
      HOLD_COST (FactoryLoc, CustomerLoc, Hold_Cost)
      FIXED_COST (FactoryLoc, CustomerLoc, Fixed_Cost)
      SHIP_QTY  (FactoryLoc, CustomerLoc, Ship_Qty)
      ORDER_QTY (FactoryLoc, CustomerLoc, Order_Qty)
      Total_Cost = Total_Trans_Cost + Total_Ship_Cost
      Total_Trans_Cost = + of (Trans_Cost)
      Trans_Cost       = (Unit_Trans_Cost) * (Ship_Qty)
      Total_Ship_Cost = + of (Shipment_Cost)
      Shipment_Cost    = (Receiving_Cost) + (Carrying_Cost)
      Receiving_Cost   = (Receiving_Freq) * (Fixed_Cost)
      Receiving_Freq   = (Ship_Qty) / (Order_Qty)
      Carrying_Cost    = (Hold_Cost) * (Order_Qty) / 2
      QTY_Supplied   = + of (Ship_Qty) by (FactoryLoc)
      QTY_Received   = + of (Ship_Qty) by (CustomerLoc)
      Meet = (Qty_Supplied ≤ Supply) AND
             (Qty_Received = Demand) AND
             (Ship_Qty ≥ 0)
II. The EDB:  omitted.
```

CHAPTER 6

DESIGN ISSUES OF AN IDB

Relational data base theory, the foundation of a functional data base, provides design methodology and evaluation criteria regarding the design of an IDB. Primitive relations with desirable properties can be designed by applying normalization algorithms (Hawryszkiewycz 1984), one of the major contributions of relational data base theory. Furthermore, a comparison is made between expressions used to defining virtual numeric attributes and those used in a relational language. Definitions of virtual numerical attributes are actually embedded data retrieval statements using operations of a simple, flexible, and powerful relational language.

Primitive Relations in Normal Forms

A primitive relation in an IDB is restricted to an elementary form. Each primitive relation in elementary form corresponds to a generalized array used in the MMSs for MP DSMs to maintain values and variables referenced in a DSM. However, the restriction of elementary relations is not necessary as far as design of an IDB is concerned.

Normal-form Relations and the E-R Model

One of the main contributions of relational data base theory is the use of normal forms for data representation. The general objective of normal-form relations is to reduce data redundancy, and hence to avoid certain problems over addition, deletion, or update operations (Date 1985).

There are numerous normal forms (Hawryszkiewycz 1984) in the field of relational data base. The three best known are the first, second, and third normal forms. The higher level a normal form is, the more desirable it is. A relation is in first normal form if it contains atomic values only. A relation is in second normal form if every nonkey attribute is fully functionally dependent on each relation identifier. A relation is in third normal form if it is in second normal form and the nonkey attributes are mutually independent. Normalization algorithms furnish rules and guidelines to systematically reduce a relation to a collection of relations that are equivalent to the original and yet in some normal form preferable to it.

Regarding the design of a relational data base, the E-R model provides a handier way to obtain normal-form relations than normalization algorithms. As Chen (1976) put it,

> By using the E-R model, data can be arranged in a form similar to third-normal-form relations but with clear semantic meaning. It is interesting to note that the decomposition approach for normalizing relations may be viewed as a bottom-up approach in data base design. . . . The E-R model adopts a top-down approach, utilizing the semantic information to organize data in entity/relationship relations.

Conversion of an E-R model to a relational data base has been discussed by several researchers. Usually, each entity or relationship type is directly converted to an entity or relationship relation. Nevertheless, such a simple conversion does not always generate normal-form relations. To ensure normal-form relations resulting from converting an E-R model, normal forms for E-R diagrams have been proposed (Chung, Nakamura, and Chen 1981; Ling 1985). An algorithm is also presented to translate a normal-form E-R diagram into a relational data base (Ling 1985).

Normalizing Primitive Relations

So far, primitive relations in an IDB are restricted to be in elementary form, i.e., each relation contains only one numeric attribute and the identifier. One advantage of elementary relations is allowing flexible inferencing mechanisms to support decision making (Lee 1985). Another advantage of elementary relations is preventing the corresponding EDB from creating anomalies

following addition, deletion, or update operations because each elementary relation is in third normal form.

An elementary relation is in first normal form since it contains only atomic values of a numeric attribute and its identifer. An elementary relation is also in second normal form because the numeric attribute must be fully functionally dependent on the identifier. Finally, an elementary relation is in third normal form due to the fact that it contains only one nonkey attribute, that is the numeric attribute.

Despite of the advantages, elementary relations can be relaxed by applying the concept of normal forms. Design of normal-form relations in an IDB is not as complicated and difficult as in an ordinary relational data base. Following the trend of data base design, the E-R model can be used to develop primitive relations of an IDB. Moreover, an ERD of a DSM can be easily converted into a set of normal-form relations in an IDB by applying the simple conversion rule.

Usually, it is because of the existence of multi-valued attributes that the simple conversion rule fails to produce normal-form relations from an ERD. Yet, this is not the case as far as primitive relations of an IDB are concerned. In the mathematical representation of a DSM, indices are used to denote similar parameters and decision variables of a DSM. Hence, each indexed parameter denotes

a specific value, not a set of values, referenced by the DSM, and each indexed decision variable denotes an unknown value to be determined by the solution of the DSM.

Identifiers of primitive relations are to be chosen in such a way that they can serve the role of indices or subscripts. Consequently, in a primitive relation, a numeric attribute has only one value corresponding to a value of its identifier. That means, a primitive relation does not contain any multivalued attribute. Since no primitive relation in an IDB contains multivalued attributes, the simple conversion rule suffices for converting an ERD of a DSM to a set of normal-form relations in an IDB.

Following the simple conversion rule, primitive relations resulting from converting an ERD of a DSM are different from elementary relations. One obvious difference is that numeric attributes, having an identical identifier, are maintained in a primitive relation, rather than in separate relations. Thus, values of the identifer are stored only once in a bigger relation, not repeatedly in several relations.

To design normal-form primitive relations in an IDB, one step needs to be added to the design procedures of a functional DSM: generate a relation by combining elementary relations with an identical identifier. One example is the normalized primitive relations and the EDB

of the tariff rates DSM shown in Table 34. The total number of primitive relations is reduced from 12 in Table 17 to 4 in Table 34.

Table 34.--The Normalized Primitive Relations and the EDB of the Tariff Rates DSM

I. The Normalized Primitive Relations in the IDB:

UPLOAD (Upload)
PERIOD (Period, Hours, Load_Demand)
GENERATOR (Generator, Gu_Available, Min_Level, Max_Level, Costph, Excostph, Scostpu)
WORKING (Period, Generator, Number, Nst, Out)

II. The EDB:

PERIOD

(Period,	Hours,	Load_Demand)
1	6	15000
2	3	30000
3	6	25000
4	3	40000
5	6	27000

UPLOAD (Upload)
1.15

GENERATOR

Gene- (rator,	Gu_ Available,	Min_ Level,	Max_ Level,	Costph,	Ex_ Costph,	Scostpu)
G1	12	850	2000	1000	2.0	2000
G2	10	1250	1750	2600	1.3	1000
G3	5	1500	4000	3000	3.0	500

WORKING

(Period,	Generator,	Number,	Nst,	Out)	
1	1	X_{11}	Y_{11}	Z_{11}	
2	1	X_{21}	Y_{21}	Z_{21}	
3	1	X_{31}	Y_{31}	Z_{31}	
4	1	X_{41}	Y_{41}	Z_{41}	Corresponding
5	1	X_{51}	Y_{51}	Z_{51}	Decision
1	2	X_{12}	Y_{12}	Z_{12}	Variables
2	2	X_{22}	Y_{22}	Z_{22}	
3	2	X_{32}	Y_{32}	Z_{32}	
4	2	X_{42}	Y_{42}	Z_{42}	
5	2	X_{52}	Y_{52}	Z_{52}	
1	3	X_{13}	Y_{13}	Z_{13}	
2	3	X_{23}	Y_{23}	Z_{23}	
3	3	X_{33}	Y_{33}	Z_{33}	
4	3	X_{43}	Y_{43}	Z_{43}	
5	3	X_{53}	Y_{53}	Z_{53}	

Definitions of Virtual Attributes

As described, definitions of virtual numeric attributes can be viewed as embedded data retrieval statements using operations of a relational language. A comparison between operations of domain algebra with those of SQL is listed in Table 35. SQL is an implemented relational language (Lans 1988). The basic construct of SQL is a mapping, whose syntax takes the form,

```
SELECT   <attribute>
FROM     <relation>
WHERE    <condition clause>
```

The output from a mapping is a set of values. The values are chosen by selecting each relation row that satisfies the condition clause. The value of the attribute of each such selected row becomes part of the output.

Table 35.--Comparison of Domain Algebra with SQL

Operations of Domain Algebra used in IDB	SQL
1. Scalar operation.	Retrieval of computed values.
2. Simple reduction.	the standard functions SUM, MAX, and MIN.
3. Equivalence reduction.	GROUP BY.
4. Simple functional mapping.	-.
5. Partial functional mapping.	-.

Basically, there is a major difference between the definition language used in an IDB and SQL. Expressions in definitions of virtual numeric attributes are for the purpose of describing the mathematical structure of a DSM; while those of SQL provide flexible ways of retrieving data. In other words, an expression used to define a virtual numerical attribute may not be evaluated because it may involve one or more unknown decision variables. Yet, an SQL expression is always ready to be evaluated because values of its operand attributes are stored in a relational data base.

Scalar Operation versus Retrieval of Computed Values

A definition using a scalar operation is similar to a retrieval of computed values. For example, the definition of "Trans_Cost" in the transportation DSM,

```
Trans_Cost = Unit_Trans_Cost * Ship_Qty
```

is similar to the data retrieval statement,

```
SELECT    FactoryLoc, CustomerLoc,
          Unit_Trans_Cost * Ship_Qty
FROM      SHIPMENT
```

except that the data retrieval statement cannot be evaluated because "Ship_Qty" is a variable attribute. Besides, the expression, Unit_Trans_Cost * Ship_Qty, is not given a meaningful name in the data retrieval statement.

Simple Reduction versus Standard Functions of SQL

A definition using a simple reduction is similar to a retrieval statement using a standard function such as SUM, MAX, or MIN of SQL. For example, the definition of "Total_Trans_Cost" in the transportation DSM,

Total_Trans_Cost = + of (Trans_Cost)

is similar to the data retrieval statement,

```
SELECT   SUM(Trans_Cost)
FROM     SHIPMENT
```

assuming that the virtual attribute "Trans_Cost" is included in the relation SHIPMENT. Again, the data retrieval statement cannot be evaluated because attribute "Trans_Cost" is mathematically dependent on a variable attribute "Ship_Qty". Also, the expression, SUM(Trans_Cost), is not given a mnemonic name in the data retrieval statement as in the definition.

Equivalence Reduction versus the Clause GROUP BY

A definition using an equivalence reduction is similar to a data retrieval statement using the clause GROUP BY. For example, the definition of "Qty_Supplied" in the transportation DSM,

Qty_Supplied = + of (Ship_Qty) by (FactoryLoc)

resembles the data retrieval statement,

```
SELECT    FactoryLoc, Ship_Qty
FROM      SHIP_QTY
GROUP     BY FactoryLoc
```

except that the data retrieval statement cannot be evaluated because attribute "Ship_Qty" is a variable attribute. Note that the operation "+" is not explicitly requested in the data retrieval statement; it is implied by the GROUP BY clause. Hence, an equivalence reduction using an operation other than "+" cannot be expressed in a similar data retrieval statement using the clause GROUP BY.

Functional Mapping versus the clause ORDER BY

Though it seems that a definitional expression using simple functional mapping is similar to a data retrieval statement using the clause ORDER BY, they are indeed different. The result of a simple functional mapping operation is dependent on the order of tuples in the operand relation. It is the ordering attributes that specify the order for the tuples of the operand relation. Nonetheless, the ORDER BY clause of SQL simply sorts and presents the output of a retrieval based on values of ordering attributes. The clause does not change the contents of the output; hence it is not a means for handling relationships among tuples of a relation.

Suppose that the annual sales stored in the primitive relation SALES are as follows.

SALES	(Year,	Sales)
	1984	150,000
	1985	175,000
	1986	200,000
	1987	210,000
	1988	225,000

Cumulative annual sales is the sum of annual sales according to the order of "Year".

Cum_Sales = + of (Sales) order (Year)

CUM_SALES	(Year,	Cum_Sales)
	1984	150,000
	1985	325,000
	1986	525,000
	1987	735,000
	1988	960,000

A plausible similar data retrieval statement is as below.

```
SELECT   Year, Sales
FROM     SALES
ORDER    BY Year
```

However, the result of the retrieval is the same as relation SALES, not relation CUM_SALES.

Partial Functional Mapping versus GROUP BY and ORDER BY

Since a simple functional mapping is in no way similar to a data retrieval statement using the clause ORDER BY, partial functional mapping is, of course, not similar to a data retrieval statement using clauses GROUP BY and ORDER BY together. There is no equivalent data retrieval statement in SQL for partial functional mapping.

CHAPTER 7

TRANSLATING A FUNCTIONAL DSM

An important issue of MMSs is DSM translation. The functional MMS must be capable of translating an instance of a functional DSM into a conventional format which can be directly entered to a computer and solved by a solution procedure. Because an IDB contains dimension-free descriptions of a functional DSM, the first step of the translation is to generate all the similar decision variables with appropriate indices. Then the decision variables as well as data of parameter attributes are used to interpret definitions of virtual attributes and to generate mathematical expressions of the DSM.

A computer program TRANSLATOR has been developed to translate an instance of a functional DSM into the corresponding mathematical format. The discussion of TRANSLATOR includes the input files and assumptions, the interpretation of virtual numeric attributes, and the overall description of TRANSLATOR. Most of the examples used to explain TRANSLATOR are drawn from the standard transportation DSM.

TRANSLATOR is written in Turbo PROLOG Version 2.0 on IBM Personal Computer AT. As mentioned, PROLOG is the

best-known of the logic programming languages. The theoretical foundation of PROLOG is the predicate calculus. In PROLOG a predicate is like a subroutine in a high-level programming language. The complete PROLOG program of TRANSLATOR is presented in Appendix B.

The Input Files and Assumptions of TRANSLATOR

TRANSLATOR reads three input files: a functional model base, an IDB and EDB of a DSM, interprets definitions of virtual numeric attributes, and generates mathematical expressions of the DSM. For the sake of simplicity, all input files are saved in a PROLOG readable format and can be read into the working data base of TURBO PROLOG using the built-in predicate "consult".

The Input File Functional Model Base

A functional model base input to TRANSLATOR contains classes of DSMs which are represented by the predicate "dsm". A DSM predicate has five arguments: a DSM name, the input list, the output list, the objective list, and the logical attribute. Both of the DSM name and logical attribute are denoted as symbols in TURBO PROLOG. The input and output lists of the predicate "dsm" are expressed as lists of symbols. The objective list is made up of a functor, "obj_func", each of which has two arguments: an optimization indicator (max or min) and a symbol for the objective attribute. An example of a

functional model base is shown in Figure 31.

```
dsm("transport", ["unit_trans_cost", "supply",
    "demand"], ["ship_qty"],
    [obj_func(min,"total_trans_cost")], "meet").
dsm("prodmix", ["unit_price", "resource_unit_cost",
    "unit_use", "resource_available"],
    "prod_units"], [obj_func(max,"total_profit")],
    "enough").
dsm("feedmix", ["unit_cost", "analysis",
    "min_nutr"], ["qty"],
    [obj_func(min,"total_material_cost")],
    "exceed").
dsm("plant", ["prod_unit_profit", "unit_use",
    "pcs_capacity", "need", "available"],
    ["prod_qty"], [obj_func(max,"total_profit")],
    "enough").
dsm("eoq", ["demand_rate", "hold_cost",
    "fixed_cost"], ["order_qty"], [obj_func(min,
    "total_order_cost")], "").
dsm("tariff", ["excostph","hours","min_level",
    "costph", "scostpu", "max_level", "upload",
    "load_demand", "gu_available"],
    ["number","nst","out"],
    [obj_func(min,"total_op_cost")], all")
dsm("integrate", ["unit_trans_cost", "supply",
    "demand", "hold_cost", "fixed_cost"],
    ["ship_qty","order_qty"],
    [obj_func(min,"total_cost")], "meet").
```

Figure 31. The Input File of a Functional Model Base

The Input File IDB

The input file IDB is composed of definitions of attributes using the predicates "relation", "ldefn", and "defn". The predicate "relation" is used to declare a primitive relation, the predicate "ldefn" is for defining a logical attribute, and the predicate "defn" is to define a virtual numeric attribute.

The predicate "relation" contains three arguments: the name of a primitive relation, a list of symbolic

attributes as the identifier, and a list of primitive numeric attributes. The predicate "ldefn" has two arguments: a logical attribute name and a list of conditions. Each condition is expressed as a functor "c" with a binary operator and the names of two numeric attributes to be compared. The binary operators include "lt" (less than), "le" (less than or equal to), "eq" (equal to), "gt" (greater than), and "ge" (greater than or equal to).

The predicate "defn" is with two arguments: the name of a virtual numeric attribute and an expression to define the attribute. An expression can use any one of the five operations: scalar operation, simple reduction, equivalence reduction, functional mapping, and partial functional mapping. Formats of expressions using each operation are shown in Table 36.

In an expression, the functor "r" is to denote an operand attribute, and thus contains the name of a numeric attribute as the only argument. An operand attribute must be contained in a primitive relation or defined as a virtual attribute. The other functor "o" is to specify a mathematical operator and has an argument, a permissible operator. The functor "s" is to denote a reserved word such as "of", "by", or "order". Finally, the functor "l" denotes a list of control or ordering attributes. An example of the input file IDB is presented in Figure 32

which is drawn from the transportation DSM.

Table 36.--Formats of Expressions Used in the Input File IDB to Define Virtual Numeric Attributes

Operations	Original Format	Input Format (see notes)
Scalar operation	<expression>	[r<nv1>), o(<operator>), r(<nv2>)]
Simple reduction	<op> of (<na>)	[o(<operator>), s(of), r(<nv>)]
Equivalence reduction	<op> of (<na>) by (<ca>)	[o(<operator>), s(of), r(<nv>), s(by), l(<cv>)]
Functional mapping	<op> of (<na>) order (<oa>)	[o(<operator>), s(of), r(<nv>), s(order), l(<ov>)]
Partial functional mapping	<op> of (<na>) order (<oa>) by (<ca>)	[o(<operator>), s(of), r(<nv>), s(order), l(<ov>), s(by), l(<cv>)]

Notes: 1. The functor "r" contains an argument, the name of a numeric attribute; <nv1>, <nv2>, and <nv> denote numeric attributes.
2. The functor "o" contains an argument, a mathematical operator; <operator> is a permissible operator.
3. The functor "s" denotes a reserved word used in an expression such as "of", "by", or "order".
4. The functor "l" contains an argument, a list of control or ordering attributes; <cv> denotes a list of control attributes, and <ov> denotes a list of ordering attributes.

```
relation("supply",["factoryloc"],["supply"])
relation("demand",["customerloc"],["demand"])
relation("unit_trans_cost", ["factoryloc",
    "customerloc"], ["unit_trans_cost"])
relation("ship_qty",["factoryloc","customerloc"],
    ["ship_qty"])
ldefn("meet",[c("qty_supplied",le,"supply"),
    c("qty_received",eq,"demand"),
    c("ship_qty",ge,"zero")])
defn("qty_supplied", [o("+"), s("of"),
    r("ship_qty"),s("by"), l(["factoryloc"])])
defn("qty_received",[o("+"),s("of"), r("ship_qty"),
    s("by"), l(["customerloc"])])
defn("trans_cost",[r("unit_cost"), o("*"),
    r("qty")])
defn("total_cost",[o("+"), s("of"),
    r("trans_cost")])
```

Figure 32. The Input File IDB for the Transportation DSM

The Input File EDB

An EDB input to TRANSLATOR consists of rows of data for primitive relations declared in the IDB. Each row of data is represented by the predicate "tuple" with three arguments: the name of a primitive relation, a list of data for the symbolic attributes, and a list of data for the primitive numeric attributes. Data of attributes are in the same order as the attributes declared in the corresponding predicate "relation". Figure 33 depicts an input file EDB for the primitive relations declared in Figure 32. Note that there are five dummy rows listed for relation SHIP_QTY to specify all the similar decision variables of the transportation DSM.

```
tuple("supply",["dallas"],["20000"])
tuple("supply",["chicago"],["42000"])
tuple("demand",["pittsburgh"],["25000"])
tuple("demand",["atlanta"],["15000"])
tuple("demand",["cleveland"],["22000"])
tuple("unit_trans_cost",["dallas","pittsburgh"],
    ["23.50"])
tuple("unit_trans_cost",["dallas","atlanta"],
    ["17.75"])
tuple("unit_trans_cost",["dallas","cleveland"],
    ["32.45"])
tuple("unit_trans_cost",["chicago","pittsburgh"],
    ["7.60"])
tuple("unit_trans_cost",["chicago","cleveland"],
    ["25.75"])
tuple("ship_qty",["dallas","pittsburgh"],[])
tuple("ship_qty",["dallas","atlanta"],[])
tuple("ship_qty",["dallas","cleveland"],[])
tuple("ship_qty",["chicago","pittsburgh"],[])
tuple("ship_qty",["chicago","cleveland"],[])
```

Figure 33. The Input File EDB for the Transportation DSM

The Assumptions of TRANSLATOR

TRANSLATOR assumes that the information in the three input files are correct and consistent. Furthermore, TRANSLATOR assumes that no embedded DSM predicate is used in the input file IDB to define a virtual attribute. A primitive relation declared in the input file IDB is restricted to be in elementary form; thus, a primite numeric attribute is with the same name as the corresponding primitive relation.

Interpreting Virtual Numeric Attributes

The core of TRANSLATOR is the interpretation of virtual numeric attributes defined in various formats. Before a numeric attribute is interpreted, it is matched

with the one-place predicate "interpret" to determine whether the numeric attribute needs further interpretation. The rules and the corresponding PROLOG statements of the one-place predicate "interpret" are presented in Figure 34. The matching of the one-place predicate "interpret" succeeds if the numeric attribute is primitive or has already been interpreted earlier. Otherwise, the expression used to define the numeric attribute is further interpreted using the three-place predicate "interpret". The three-place predicate "interpret" matches with an expression in one of the five formats listed in Table 36.

```
interpret(Attr) :- interpreted(Attr), !.
interpret(Attr) :- defn(Attr, Exp),
                   interpret(_, Attr, Exp).
interpreted(Attr) :- relation(Attr, _, _).
```

Figure 34. Rules and the PROLOG Statements of Interpreting a Numeric Attribute

Five rules are specified for the three-place predicate "interpret" to handle the five different formats of an expression. The correct rule to interpret an expression is chosen automatically by the unification mechanism of PROLOG. In general, the three-place predicate "interpret" has three arguments. The first one indicating the type of operation used in the expression is simply for reference. The second argument contains the name of the virtual numeric attribute which is defined by

the expression, the third argument.

Interpreting Scalar Operation

For simplicity, TRANSLATOR interprets expressions using a scalar operation in a limited format (Figure 35). The simplified format is indeed not very restrictive because operand attributes are not confined to primitive ones. In other words, a complex expression can be equivalently written as a simple one using virtual numeric attributes as operand attributes which are further defined by other expressions. The difference is that, using the limited format, more virtual numeric attributes may have to be defined as intermediate attributes.

```
<expression> ::= <na> + <na> | <na> - <na> |
                 <na> * <na> | <na> / <na>
```

where <na> denotes a numeric attribute.

Figure 35. Simplified Format of Scalar Operation Interpreted by TRANSLATOR

The first rule of the three-place predicate "interpret" is to interpret an expression using a scalar operation. Figure 36 presents the rule and corresponding PROLOG statements. A simplified scalar expression is interpreted in five steps. First, both operand attributes are interpreted. Second, a temporary relation is generated and added to the IDB. The temporary relation contains both operand attributes, and is identified by the

union of their identifiers. In addition, tuples of the temporary relation are generated from tuples of the operand relations.

Rule: The interpretation of an expression having the format "r(Attr1), o(Op), r(Attr2)" includes
1. interpreting the first operand attribute Attr1,
2. interpreting the second operand attribute Attr2,
3. producing a temporary relation New_Rela with compatible Attr1 and Attr2 and adding New_Rela to the IDB,
4. computing tuples of New_Rela by applying "Op" on data pairs of Attr1 and Attr2,
5. deleting New_Rela from the IDB, and
6. adding the resulting relation with the appropriate name and identifying attributes to the IDB.

```
interpret (1, Va, [r(Attr1), o(Op), r(Attr2)]) :-
   interpret (Attr1),
   interpret (Attr2),
   product (Attr1, Attr2, New_Rela),
   cmp (Attr1, Op, Attr2, New_Rela, Va),
   retract (relation(New_Rela, Key, _), idb),
   assertz (relation(Va, Key, [Va]), idb).
```

Figure 36. The Rule and PROLOG Statements to Interpret a Simplified Scalar Expression

Third, apply the operator in the expression to the data pair in each tuple of the temporary relation, add a tuple of the virtual relation with the result to the EDB and delete the tuple of the temporary relation from the EDB. Fourth, delete the temporary relation from the IDB. Finally, add the virtual relation with the appropriate name and identifier to the IDB.

Interpreting Simple Reduction

TRANSLATOR interprets an expression using simple reduction by collecting data of the operand attribute in a temporary list and applying the operator to all the data in the list. Tuples of a relation can be easily collected in a list using the built-in predicate "findall". The permissible operator includes addition ("+") and multiplication ("*").

The second rule of the three-place predicate "interpret" is to interpret a simple reduction (Figure 37). The interpretation of a simple reduction consists of four steps. First, interpret the operand attribute. Second, collect all the data of the operand attribute from the EDB and construct a temporary data list. Third, apply the operator in the expression to reduce all the data in the temporary list to a single value and store the result in a temporary variable. Fourth, add the resulting virtual relation with the appropriate name to the IDB and a tuple with the result to the EDB.

Rule: The interpretation of an expression matching the format "o(Op), s(of), r(Attr)" includes
1. interpreting the operand attribute Attr,
2. constructing a temporary list containing data of the operand attribute from tuples of the operand relation,
3. applying the operator to data in the temporary list and storing the result in Result,
4. adding the reduced relation with the appropriate name to the IDB, and
5. adding a tuple with Result to the EDB.

```
interpret (2, Va, [o(Op), s(of), r(Attr)]) :-
     interpret(Attr),
     findall (Data, tuple(Attr, _, [Data]), Data_List),
     sreduce(Va, Op, Attr, Data_List, Result),
     assertz(relation(Va, [], [Va]), idb),
     assertz(tuple(Va, [], [Result]), edb).
```

Figure 37. The Rule and PROLOG Statements to Interpret a Simple Reduction

Interpreting Equivalence Reduction

As described, an equivalence reduction is like a simple reduction except it produces different results for different groups of tuples of the operand relations. Each group of tuples is characterized by the same value for a list of control attributes. Similar to simple reduction, the permissible operator includes addition ("+") and multiplication ("*").

The third rule of the three-place predicate "interpret" interprets an equivalence reduction (Figure 38). TRANSLATOR interprets an equivalence operation in seven steps. First, the operand attribute is interpreted. Second, for every control attribute, find its relative position in the identifier of the operand relation and

Rule: The interpretation of an expression having the format "o(Op), s(of), r(Attr), s(by), l(Ctrl)" includes

1. interpreting the operand attribute Attr,
2. for every control attribute in Ctrl, finding its relative position in the identifier KeyA of the operand relation and storing the positions in a temporary list C_Seq,
3. constructing another temporary list List containing data of the operand attribute and its identifier from tuples of the operand relation,
4. forming a reduced list CList by retaining in List only data of the operand attribute and the control attributes based on C_Seq.
5. sorting CList using values of the control attributes as the sorting key,
6. generating a tuple of the virtual relation by applying the operator to each group of data in CList which have identical values for the control attributes, and
7. adding the virtual relation with the appropriate name and with the control attributes as the identifier to the IDB.

```
interpret (3, Va, [o(Op), s(of), r(Attr), s(by), l(Ctrl)])
:-   interpret(Attr),
     relation (Attr, KeyA, _),
     seq (KeyA, Ctrl, C_Seq),
     findall (Datapair, datapair(Attr, Datapair), List),
     modify3(List, CList, C_Seq),
     my_sort(CList, [H|Rest], []),
     ereduce (Attr, Va, Op, H, Rest),
     assertz(relation(Va, Ctrl, [Va]), idb).
```

Figure 38. The Rule and PROLOG Statements to Interpret an Equivalence Reduction

store their positions in a temporary index list. Third, construct a temporary data list by including data of the operand attribute and its identifier from tuples of the operand relation. Fourth, modify the temporary data list by retaining data for the operand attribute and the control attributes using the temporary index list. Fifth, sort the data list based on values of the control

attributes. Sixth, for each group of data in the list which have identical values for the control attributes, produce a tuple of the virtual relation by applying the operator to all the data in each group. Seventh, add the virtual relation to the IDB using control attributes as the identifier.

Interpreting Functional Mapping

Simple functional mapping provides a means of computing a virtual relation based on the order specified by values of ordering attributes. Operators of a functional mapping incorporated in TRANSLATOR include addition ("+") and multiplication ("*").

The fourth rule of the three-place predicate "interpret" interprets a functional mapping. The rule and PROLOG statements are presented in Figure 39. First, interpret the operand attribute. Second, for every ordering attribute, find its relative position in the identifier of the operand relation and store their positions in a temporary index list. Third, construct a temporary data list containing data of the operand attribute and its identifier from tuples of the operand relation. Fourth, modify the temporary data list by retaining data for the operand attribute and the ordering attributes. Fifth, sort the data list based on values of the ordering attributes. Sixth, for each different value of the ordering attributes, produce a tuple of the virtual

relation by applying the operator to tuples having less or equal values. Seventh, add the virtual relation to the IDB using ordering attributes as the identifier.

Rule: The interpretation of an expression matching the format "o(Op), s(of), r(Attr), s(order), l(Ord)" includes
1. interpreting the operand attribute Attr,
2. for every ordering attribute in Ord, finding its relative position in the identifier KeyA of the operand relation and storing the positions in a temporary index list Seq,
3. constructing a temporary data list List containing data of the operand attribute and its identifier from tuples of the operand relation,
4. forming a reduced list OList by retaining in List only data of the operand attribute and the control attributes based on Seq.
5. sorting OList using values of the ordering attributes as the sorting key,
6. producing a tuple of the virtual relation for each different value of the ordering attributes by applying the operator to data in OList of which the values are not greater than the value of the ordering attributes, and
7. adding the virtual relation with the appropriate name and with the ordering attributes as the identifier to the IDB.

```
interpret (4, Va, [o(Op), s(of), r(Attr), s(order),
l(Ord)]) :-
     interpret(Attr),
     relation (Attr, KeyA, _),
     seq (KeyA, Ord, Seq),
     findall (Datapair, datapair(Attr, Datapair), List),
     modify3(List, OList, Seq),
     my_sort(OList, [H|Rest], []),
     freduce (Attr, Va, Op, H, Rest),
     assertz(relation(Va, Ord, [Va]), idb).
```

Figure 39. The Rule and PROLOG Statements to Interpret a Functional Mapping

Interpreting a Partial Functional Mapping

As an equivalence reduction is to a simple reduction, a partial functional mapping extends functional mapping in a similar way. Similar to functional mapping, operators of a partial functional mapping incorporated in TRANSLATOR include addition ("+") and multiplication ("*").

The last rule of the three-place predicate "interpret" is for interpreting a partial functional mapping. The rule and PROLOG statements to interpret a partial functional mapping are shown in Figure 40. First, the operand attribute is interpreted. Second, for every control attribute, find its relative position in the identifier of the operand relation and store their positions in a temporary index list. Third, for every ordering attribute, find its relative position in the identifier of the operand relation and store their positions in another temporary index list. Fourth, collect data of the operand attribute and its identifier from tuples of the operand relation and construct a temporary data list. Fifth, modify the temporary data list by retaining data for the operand attribute, control and ordering attributes. Sixth, sort the data list based on values of the control and ordering attributes. Seventh, generate a tuple of the virtual relation for each

Rule: The interpretation of an expression having the format "o(Op), s(of), r(Attr), s(order),l(Ord), s(by), l(Ctrl)" includes

1. interpreting the operand attribute Attr,
2. for every control attribute in Ctrl, finding its relative position in the identifier KeyA of the operand relation and storing the positions in a list C_Seq,
3. for every ordering attribute in Ord, also finding its relative position in KeyA of the operand relation and storing the positions in a list O_Seq,
4. constructing a data list List containing data of the operand attribute and its identifier from tuples of the operand relation,
5. forming a reduced list NewList by retaining only data of the operand attribute, the control attributes and the ordering attributes based on lists C_Seq and O_Seq, respectively,
6. sorting NewList using values of the control and ordering attributes together as the sorting key,
7. generating a tuple of the virtual relation by applying the operator to data in NewList according to the sequence determined by values of the ordering attributes within each group of data which have identical values for the control attributes, and
8. adding the virtual relation with the appropriate name and with the control and ordering attributes together as the identifier to the IDB.

```
interpret (5, Va, [o(Op),s(of),r(Attr), s(order), l(Ord),
s(by), l(Ctrl)]) :-
      interpret(Attr),
      relation (Attr, KeyA, _),
      seq (KeyA, Ctrl, C_Seq),
      seq (KeyA, Ord, O_Seq),
      findall (Datapair, datapair(Attr, Datapair), List),
      modify5 (List, NewList, C_Seq, O_Seq),
      my_sort2 (NewList, [H|Rest], []),
      pfreduce (Attr, Va, Op, H, Rest),
      append (Ctrl, Ord, KeyA2),
      assertz(relation(Va, KeyA2, [Va]), idb).
```

Figure 40. The Rule and PROLOG Statements to Interpret a Partial Functional Mapping

different value of the ordering attributes by applying the operator to tuples having less or equal values within each

group of data which have identical values for the control attributes. Finally, add the virtual relation using the control and ordering attributes as the identifier to the IDB.

The TRANSLATOR Program

TRANSLATOR is developed in a conversational mode. After invoking TRANSLATOR, a user is requested to provide five names: the file name of a functional model base, the name of a functional DSM, the file name of the corresponding IDB, the file name of an EDB, and an output file name. The file names must start with a letter and contain characters or numbers. A check is made for the existence of the functional DSM and files. A user needs to re-enter a name if any error is detected. If there is no error, the program starts translating the functional DSM and displays the messages along the process of translation. The process of translation is accomplished in stages (Figure 41).

Stage 1: Generating the decision variables.
Stage 2: Translating objective functions.
Stage 3: Translating constraints.
Stage 4: Generating the mathematical DSM.

Figure 41. Staged Development of TRANSLATOR

After translating a functional DSM, TRANSLATOR removes all the temporary facts added to the working data base of TURBO PROLOG during the translation process.

Then, the user is asked whether he or she wants to translate another functional DSM. The user can either enter "y" and start the translation of another functional DSM or press "n" and terminate the program.

Generating Decision Variables

Due to the dimension-independent nature of a functional DSM, the first stage of TRANSLATOR is to generate all the similar decision variables with appropriate indices. Decision variables are generated using the predicate "dv_gen" (Figure 42).

The predicate "dv_gen" is with one argument, a list of variable attributes. It is a tail recursive predicate. In other words, the predicate generates similar decision variables for the first variable attribute in the list each time and calls itself with the list of the remaining variable attributes until the list is exhausted.

The generation of decision variables for a variable attribute includes two steps. First, a fact using the one-place predicate "unknown" with the variable attribute is added to the working data base to keep a record of unknown numeric attributes. Second, each dummy tuple of the variable attribute in the EDB is replaced by the one with an appropriately indexed decision variable.

Rule: The generation of decision variables is completed when the list of variable attributes is empty.
Rule: The generation of decision variables is completed if
1. a fact using the predicate "unknown" with the first variable attribute is added to the working data base,
2. every dummy tuple of the head variable attribute in the EDB is replaced by the one with an appropriately subscripted decision variable, and
3. generating decision variables for the remaining variable attributes in the list.

```
dv_gen([]).
dv_gen([Dv|Dvs]) :-
     assertz(unknown(Dv), workbase),
     dv_gen1(Dv),
     dv_gen(Dvs).

dv_gen1(Dv) :-
     retract (tuple(Dv, X, []), edb),
     gensym(Dv, Dv_new),
     assertz (tuple(Dv, X, [Dv_new]), edb),
     fail.
dv_gen1(_).
```

Figure 42. The Rule and PROLOG Statements to Generate Decision Variables

Translating Objective Functions

Translation of a list objective attributes is accomplished using the predicate "obj_gen" (Figure 43) which is also tail recursive. That is, the predicate "obj_gen" translates the first objective attribute in the list and then calls itself with the list of the remaining objective attributes until the list is exhausted. The predicate "obj_gen" is with one argument, a list of objective attributes.

The translation of an objective attribute consists of two steps. First, interpret the objective attribute.

Second, add to the working data base a fact using the predicate "ofunc" with two arguments, the objective attribute with the optimization indicator to keep track of objective attributes.

Rule: The translation of a list of objective attributes is completed when the list is empty.
Rule: The translation of a list of objective attributes includes
1. interpreting the first objective attribute,
2. adding to the working data base a fact using the predicate "ofunc" with two arguments, the first objective attribute with optimization indicator, and
3. calling the predicate with the list of the remaining objective attributes.

```
obj_gen([]).
obj_gen([obj_func(Opt, Obj)|Objs]) :-
   interpret(Obj),
   assertz(ofunc(Opt, Obj)),
   obj_gen(Objs).
```

Figure 43. The Rule and PROLOG Statements to Translate a List of Objective Attributes

Translating A Logical Attribute

A logical attribute is translated by the predicate "constraint_gen" (Figure 44). The predicate "constraint_gen" has one argument, a logical attribute. The rule of the predicate says that translating a logical attribute is nothing but translating the list of conditions embodied in the definition of the logical attribute. Each condition corresponds to similar comparisons between values of two attributes.

Rule: Translating a logical attribute is the same as translating the conditions in the definition of the logical attribute.
Rule: The translation of a list of conditions is completed when the list is empty.
Rule: The translation of a list of conditions includes
1. interpreting the first attribute to be compared Attr1 in the first condition,
2. interpreting the second attribute to be compared Attr2 in the first condition,
3. making sure that Attr1 and Attr2 are comparable by checking their identifiers,
4. for each comparable tuple pair of Attr1 and Attr2, adding to the working data base a fact using the predicate "cons" with the binary operator and interpretations of Attr1 and Attr2, and
5. calling the predicate itself with the remaining conditions.

```
constraint_gen(Constraint) :-
      ldefn(Constraint, Conlist),
      constraint_gen1(Conlist).

constraint_gen1([]) :- !.
constraint_gen1([c(Attr1, Boolean, Attr2)|Conds]) :-
 interpret(Attr1),
  interpret(Attr2),
  relation (Attr1, Key1, _),  /* check comparability  */
  relation (Attr2, Key2, _),
  set_equal(Key1, Key2),
  constraint_gen2 (Boolean, Attr1, Attr2),
  constraint_gen1 (Conds).

constraint_gen2 (Boolean, Attr1, Attr2) :-
  tuple (Attr1, Key1, [Data1]),
  tuple (Attr2, Key2, [Data2]),
  set_equal(Key1, Key2),
  assertz(cons(Boolean, Data1, Data2)),
  fail.
constraint_gen2 (_, _, _).
```

Figure 44. The Rule and PROLOG Statements to Translate Logical Attribute

TRANSLATOR uses the predicate "constraint_gen1" to translate a list of the conditions each of which consists

a binary operator and two numeric attributes to be compared. The predicate "constraint_gen1" is also a tail recursive. It is satisfied when the list of conditions is empty or when all the conditions in the list are translated.

A condition is translated in three steps. First, the two numeric attributes to be compared are interpreted. Second, compare the identifiers of the two numeric attributes to ensure they are comparable. Third, for every compatible tuple pair of numeric attributes, add to the working data base a fact using the predicate "cons" with the binary operator and the interpretations of the two numeric attributes to keep track of all the constraints.

Generating the Mathematical DSM

The final stage of translating a functional DSM is generating the mathematical DSM which is handled by the predicate "convert" (Figure 45). The predicate "convert" is with one argument, the output file name. After opening the output file, TRANSLATOR writes to the output file objective functions of the functional DSM using the predicate "wrt_obj", then generates the constraints using the predicate "wrt_consnt".

TRANSLATOR generates an objective function for every fact using the predicate "ofunc" added to the working data base when objective attributes are translated.

```
Rule: The generation of the mathematical DSM includes
      1. opening the output file,
      2. generating the objective functions of the DSM,
      3. generating the constraints of the DSM, and
      4. closing the output file.
Rule: The generation of objective functions of a DSM is
      nothing but generating an objective function for
      each fact using the predicate "ofunc" in the working
      data base.
Rule: The generation of constraints of a DSM is nothing
      but generating a constraint for every fact using the
      predicate "cons" in the working data base.


convert(DsmFile) :-
   openwrite(dsmfile, DsmFile),
   writedevice(dsmfile),
   wrt_obj,
   write("subject to \n"),
   wrt_consnt,
   writedevice(screen),
   closefile(dsmfile),
   !.

/*-------------------------------------------------------*/
/* Write objective functions.                            */
wrt_obj :-
      retract(ofunc(Opt, Obj)),
      tuple(Obj, [], [ObjF]),
      write(Opt, "\t", ObjF, "\n"),
      fail.
  wrt_obj :- nl.

/*-------------------------------------------------------*/
/* Write constraints.                                    */
wrt_consnt :-
      retract(cons(Boolean, Data1, Data2)),
      logic_op(Boolean, Lop),
      write("\t", Data1, "\t", Lop, "\t", Data2, "\n"),
fail.
wrt_consnt.
```

Figure 45. The Rule and PROLOG Statements to Generate a Mathematical DSM

Similarly, TRANSLATOR generates a constraint of the DSM for every fact using the predicate "cons" added to the working data base when conditions of a logical attribute

are translated. Using the inputs depicted in Figures 31, 32, and 33, the mathematical DSM of "transport" generated by TRANSLATOR is presented in Figure 46.

```
min   23.50*ship_qty1 + 17.75*ship_qty2 + 32.45*ship_qty3
    +  7.60*ship_qty4 + 25.75*ship_qty5

subject to ship_qty4 + ship_qty5                <= 42000
           ship_qty1 + ship_qty2 + ship_qty3 <= 20000
           ship_qty2                 = 15000
           ship_qty3 + ship_qty5 = 22000
           ship_qty1 + ship_qty4 = 25000
           ship_qty1 >= 0
           ship_qty2 >= 0
           ship_qty3 >= 0
           ship_qty4 >= 0
           ship_qty5 >= 0
```

Figure 46. The Mathematical DSM for "transport" Generated by TRANSLATOR

CHAPTER 8

CONCLUSION AND FURTHER RESEARCH

Among the major components of a DSS (Figure 2b), the data management system is already widely used in commercial data processing; whereas the MMS, apart from a few notable exceptions such as PLATOFORM, is limited to research laboratories. It is certain, however, that their integration is essential for the design of a DSS. The functional approach to MMSs provides a practical solution to the integration problem and incorporates the intelligent capability of a knowledge-based MMS, while at the same time ensuring efficient and secure data management through a relational data management system. This chapter draws the conclusion from the present study and outlines plans for further research and development.

Conclusion

The functional MMS is intended to provide the two-level model management capability with all the desirable features: being knowledge-based, being flexible, independence, and being able to reflect a user's viewpoint. At the macro level, the functional MMS is based on first-order logic, which is probably the best-

developed knowledge representation methodology. At the micro level, the foundation of the functional MMS is on the relational theory which has proven its usefulness in data management. In other words, functional DSMs are directly drawn from a relational data base. Additionally, the definitional system in a functional data base provides a natural way of developing a DSM in a hierarchical manner.

To really achieve the objective, it will be necessary to develop professional quality software based on the ideas of the functional MMS, and to produce tutorial materials for DSM builders. These materials should also explain how to use the functional MMS in conjunction with conventional software for solving DSMs. The computer program TRANSLATOR is a prototype system to demonstrate the computer representation of a functional model base and functional DSM. It aims also to show that an instance of a functional DSM can be translated into a mathematical DSM which can be directly entered to a solution procedure for the solution.

Further Research and Development

The functional MMS can be further studied from the aspects of the expressive scope, the relationships among DSM predicates, IDBs and EDBs, the extensions of the functional MMS, and the implementation issues of the functional MMS.

Expressive Scope of the Functional MMS

The types of DSMs can be described by the functional approach depends on the types of operators available in the definitional system. It would be useful to study the representational scope of the functional MMS in a theoretical manner. For example, using the five operators of domain algebra, what classes of DSMs can be rendered as a functional DSM? In general, ordinary MP DSMs and network DSMs are among those which can always be expressed as a functional DSM.

Furthermore, the syntax and semantics of the definitional system could be refined to facilitate expressing mathematical expressions that are presently impossible to express. It would be useful to understand what types of operators are necessary in order to represent a particular type of DSMs using the functional approach. For example, how can a statistical relationship, rather than a functional one, among numeric attributes be expressed in an IDB? How can a virtual attribute be defined as a function of continuous time?

Relationships Among DSM Predicates, IDBs and EDBs

Besides, it would be important to study, in the functional MMS, how to manage the use of DSM predicates, IDBs and EDBs; moreover, how to support the capability of binding a DSM predicate of interest with an IDB and EDB

automatically based on their inferred correspondences. A complete description of a functional DSM is made up of a DSM predicate and the corresponding IDB and EDB. The functional MMS must allow only the use of DSM predicates, IDBs and EDBs in a consistent manner by maintaining the relationships among them. Basically, a DSM predicate must be used only with an IDB in which all the attributes appearing in the DSM predicate are defined and used appropriately. On the other hand, an IDB must be used together with an EDB which provides data for all the primitive relations containing the parameter attributes.

The Extensions of the Functional MMS

It would be also useful to study the possible extensions of the functional MMS. One possibility would be to explicitly maintain DSM relationships in the functional model base. Instead of being deduced from examining arguments of DSM predicates, DSM relationships can be explicitly represented to provide more efficient macro-level model management functions. In addition, performance of a functional model base can be improved by categorizing DSM predicates by the type, the purpose, the users, and others. Categorization of DSM predicates is an application of the classification system in knowledge-based systems.

As mentioned, the syntax and semantics of the definitional system in an IDB could be refined to

facilitate the mathematical expressions, such as statistical relationships among numeric attributes, that are presently impossible to express. By doing so, the functional MMS is extended by allowing more classes of DSMs, such as regression DSMs, to be incorporated in the functional model base.

It is also possible to extend the functional MMS by formalizing the operations of DSM manipulation functions. For example, an important operation is joining two IDBs together in such a way that equivalent attributes are merged. Another important operation of manipulating functional DSM is combining two EDBs of the same IDB in such a manner that the dimension of the functional DSM is automatically augmented.

The functional MMS can also be extended by allowing numeric attributes in an EDB to have default values or values that are specified only probabilistically. This would facilitate expressing stochastic DSMs and Monte Carlo simulations as a functional DSM. It is in the Syntel programming system (Risch et al. 1988) that probability distribution is introduced to relational tables. However, the introduction of probability distribution would considerably complicate the interpretations and evaluations of virtual numeric attributes which are defined directly or indirectly on a probabilistically-distributed primitive numeric attribute

in an IDB.

Implementation Issues of the Functional MMS

The computer implementations of the functional MMS requires two basic functions, maintaining functional DSMs and interfacing with the solution procedure management system.

Basically, the maintenance of functional DSMs is simple and straightforward. A complete functional DSM consists of a DSM predicate and a functional data base which is a relational data base expanded with a set of definitions. It is obvious that the creation and maintenance of DSM predicates is simple. Furthermore, because a functional data base, except the set of definitions, is a relational data base, it can be created and maintained using a relational language. As to the definitional system, a syntax-directed editor can be used to help users define virtual attributes.

However, a functional DSM has much more semantic content than a relational data base schema. Thus, a design challenge of the functional MMS is how to develop a schema-directed software that is simpler to use and more powerful than whatever is adopted in relational database systems. In other words, the functional MMS must enforce the consistency among the descriptions in DSM predicates, and the corresponding IDBs and EDBs.

Another design challenge of the functional MMS is to

develop the interface with the solution procedure management system. The interface can take several forms. One is to ask the user to fill out a control table whenever a solution procedure is to be invoked. A computer program, such as TRANSLATOR, can be developed to read an instance of a functional DSM and to construct the necessary inputs of the solution procedure. Another is to make the interface fully automatic by developing a knowledge-based program that can read an instance of a functional DSM and select automatically the most appropriate solution procedure according to the query posed by the user and the mathematical nature of the DSM. This is a question of DSM recognition and classification.

REFERENCES

Allen, F. W., M. E. S. Loomis, and M. V. Mannino. The integrated dictionary/dictionary system. Computing Surveys, 1982, 14 (2), 245-286.

Applegate, L. M., B. R. Konsynski, and J. F. Nunamaker. Model management systems: Design for decision support. Decision Support Systems, 1986, 2, 81-91.

Bisschop, J., and Meeraus, A. On the development of a general algebraic modeling system in a strategic planning environment. Mathematical Programming Study, October 1982, 20, 1-29.

Bhargava, H., M. Bieber, and S. O. Kimbrough. Oona, Max and the WYWWYWI principle: generalized hypertext and model management in a symbolic programming environment. In Proceedings of the Ninth International Conference on Information Systems. Minneapolis, Minn.: Society for Management Information Systems, 1988.

Blanning, R. W. A relational framework for model management in decision support systems. DSS-82 Transactions, 1982, 16-28.

_____. A relational framework for join implementation in model management. Decision Support Systems, 1985, 1, 69-82.

_____. An entity-relationship approach to model management. Decision Support Systems, 1986, 2, 65-72.

_____. A relational theory of model management. In C. W. Holsapple, and A. B. Whinston (Eds.) Decision Support Systems: Theory and Application. New York: Springer-Verlag, 1987.

Boisvert, R. F., S. E. Howe, and K. Kahaner. GAMS: A framework for the management of scientific software. ACM Transactions on Mathematical Software, December 1985, 11 (4), 313-355.

Bonczek, R. H., C. W. Holsapple, and A. B. Whinston. Foundations of Decision Support Systems. New York: Academic Press, 1981a.

_____. A generalized decision support systems using predicate calculus and network database management. Operations Research, 1981, 29 (2), 263-281.

Buchanan, B. G., and E. H. Shortliffe (Eds.) Rule Based Expert Systems. Reading, MA: Addison-Wesley, 1984.

Chang, C., and R. C. Lee. Symbolic Logic and Mechanical Theorem Proving. New York: Academic Press, 1973.

Chen, M. C., J. Fedorowicz, and L. J. Henschen. Deductive processes in databases and decision support systems. In Proceedings North Central Regional ACM 1982 Conference, Milwaukee, MI; November 1982.

Chen, P. P. The entity-relationship model -- toward a unified view of data. ACM Transactions on Data Base Systems, March 1976, 1 (1), 9-36.

Chen, Y. S. An entity-relationship approach to decision support and expert systems. Decision Support Systems, 1988, 4, 225-234.

Chen, Y. S., and J. M. Pruett. Expert systems and operational integration. In Computer-Aided Engineering Applications, Proceedings of the ASME 5th National Congress, 1987.

_____. An entity-relationship approach to the model management design process. Working Paper, Louisiana State University, July 1988.

Chung, I., F. Nakamura, and P. P. Chen. A decomposition of relations using the entity-relationship approach. In P. P. Chen (Ed.) Entity-Relationship Approach to Information Modeling and Analysis, ER Institute, 1981.

CODASYL Systems Committee. CODASYL Data Base Task Group Report. New York: ACM, 1971.

Codd, E. F. A relational model of data for large shared data banks. Communications of the ACM, June 1970, 13 (6), 377-387.

_____. Relational completeness of data base sublanguage. In Rustin R. (Ed.) Data Base Systems, Englewood Cliffs, N. J.: Prentice-Hall, 1972, 65-98.

Courtney, J. F., D. B. Paradice, and N. H. Mohammed. A knowledge-based DSS for managerial problem diagnosis. Decision Sciences, Summer 1987, 18 (3), 373-399.

Courtney, J. F. and D. B. Paradice. Database Systems for Management. St. Louis: MO, Times Mirror/Mosby College Publishing, 1988.

Date, C. J. An Introduction to Database Systems (4th Ed.), Reading, MA: Addison-Wesley, 1985.

Dhar, V., and A. Croker. Knowledge-based decision support in business: issues and a solution. IEEE Expert, Spring 1988, 53-62.

Dolk, D. R., and B. R. Konsynski. Knowledge representation for model management systems. IEEE Transactions on Software Engineering, 1984, SE-10 (6), 619-628.

Dolk, D. R. A generalized model management system for mathematical programming. ACM Transactions on Mathematical Software, June 1986, 12 (2), 96-126.

Dutta, A., and A. Basu. An artificial intelligence approach to model management in decision support systems. IEEE Computer, September 1984, 17 (9), 89-97.

Elam, J., J. Henderson, and L. Miller. Model management systems: an approach to decision support in complex organizations. In Proceedings of the First Conference on Information Systems, Chicago, IL:Society for Management Information Systems, 1980.

Elam, J. J. and B. Konsynski. Using artificial intelligence techniques to enhance the capabilities of model management systems. Decision Sciences, Summer 1987, 18 (3), 487-501.

Ellison, E. F. D., and G. Mitra. UIMP: User interface for mathematical programming. ACM Transactions on Mathematical Software, September 1982, 8 (3), 229-255.

Fedorowicz, J., and G. B. Williams. Representing modeling knowledge in an intelligent decision support system. Decision Support Systems, March 1986, 2 (1), 3-14.

Fourer, R. Modeling languages versus matrix generators for linear programming. ACM Transactions on Mathematical Software, June 1983, 9 (2), 143-183.

Geoffrion, A. M. An introduction to structured modeling. Management Science, May 1987, 35, 547-588.

_____. The formal aspects of structured modeling. Operations Research, January-February 1989, 37 (1), 30-51.

Greenberg, H. J. A functional description of ANALYZE: computer-assisted analysis system for linear programming models. ACM Transactions on Mathematical Software, March 1983, 9 (1), 18-56.

Hawryszkiewycz, I. T. Database Analysis and Design, Chicago, Science Research Associates, 1984.

Henderson, H. C. Finding synergy between decision support systems and expert systems research. Decision Sciences, Summer 1987, 18 (3), 333-349.

Jarke, M., and Y. Vassiliou. Coupling expert systems with database management systems. In W. Reitman (Ed.) Artificial Intelligence Applications for Business. Norwood, N. J.: ABLEX, 1984.

Katz, S., L. J. Risman, and M. Rodeh. A system for constructing linear programming models. IBM System Journal, 1980, 19 (4), 505-520.

Keen, P. G. W. Adaptive design for DSS. Database, Fall 1980, 12 (1) and (2), 15-25.

Keen, P. G. W., and M. S. Scott Morton. Decision Support Systems: An Organizational Perspective. Reading, MA: Addison-Wesley, 1978.

Kitzmiller, C. T. and J. S. Kowalik. Symbolic and numerical computing in knowledge-based systems. In J. S. Kowalik (Ed.) Coupling Symbolic and Numeric Computing in Expert Systems. North-Holland: Elsevier, 1986.

Klein, G. Developing model strings for model managers. Journal of Management Information Systems, 1986, 3 (2), 94-110.

Klein, G., B. Konsynski and P. Beck. A linear representation for model management in a DSS. Journal of Management Information Systems, 1985, 2 (2), 40-54.

Konsynski, B. Model management in decision support systems. In Holsapple, C. W. and A. B. Whinston, A. B. (Eds.) Data Base Management: Theory and Applications. Reidel, 1983, 131-154.

Lans, R. F. Introduction to SQL. New York: Addison Wesley, 1988.

Larson, J. A., S. B. Navathe and R. Elmasri. A theory of attribute equivalence in databases with application to schema integration. Transactions on Software Engineering, April 1989, 13 (4), 449-463.

Lee, R. M. Database inferencing for decision support. Decision Support Systems, 1985, 1, 57-68.

Lenard, M. E. Representing models as data. Journal of Management Information Systems, Spring 1986, 2 (4), 36-48.

Liang, T. P. Development of a knowledge-based model management system. Operations Research, December 1988, 36 (6), 849-863.

Liang, T. P. and C. Jones. Meta-design considerations in developing model management systems. Decision Sciences, Winter 1988, 19 (1), 72-92.

Libby, R. Accounting and Human Information Processing: Theory and Applications, Englewood Cliffs, NJ: Prentice-Hall Inc., 1981.

Ling, T. W. A normal form for entity-relationship diagrams. IEEE Proceedings of the 4th International Conference on Entity-Relationship Approach, Chicago, IL, October 28-30, 1985, 24-35.

Little, J. D. C. Models and managers: the concept of a decision calculus. Management Science, April 1970, 16 (8), B-466-485.

Mannino, M. V., B. S. Greenberg, and S. N. Hong. Knowledge representation for model libraries. In Proceedings of Twenty-First Hawaii International Conference on Systems Sciences, January 1988.

Merrett, T. H. Relational Information Systems. Reston, VA: Reston, 1984.

Mills, R. E., R. B. Fetter, and R. F. Averiall. A computer language for mathematical program formulation. Decision Sciences, 1977, 8, 427-444.

Minch, R. P., and J. R. Burns. Conceptual design of decision support systems utilizing management science models. IEEE Transactions on Systems, Man, and Cybernetics, July/August 1983, SMC-13 (4), 549-557.

Minsky, M. A framework for representing knowledge. In P. Winston (Ed.) The Psychology of Vision, New York: McGraw-Hill, 1975.

Mittra, S. S. Decision Support Systems: Techniques. New York: John Wiley & Sons, 1986.

Orman, L. An array algebra for database applications. Journal of Management Information Systems, 1985, 1 (4), 44-56.

_____. Flexible management of computational models. Decision Support Systems, 1986, 2, 225-234.

Palmer, K. H., N. K. Boudwin, H. A. Patton, A. J. Rowland, J. D. Sammes and D. M. Smith. A Model-Management Framework for Mathematical Programming. New York: John Wiley & Sons, 1984.

Paradice D. B., and J. F. Courtney. Causal and non-causal relationships and dynamic model construction in a managerial advisory system. Journal of Management Information Systems, Spring 1987, 3 (4), 39-53.

Potter, W. D., and R. P. Trueblood. Traditional, semantic, and hyper-semantic approaches to data modeling. IEEE Computer, June 1988. 53-63.

Pracht, W. E. GISMO: a visual problem-structuring and knowledge-organization tool. IEEE Transactions on Systems, Man, and Cybernetics, 1986, 16 (2), 265-270.

Rich, E. Artificial Intelligence, New York: McGraw-Hill, 1983.

Risch, T., R. Reboh, P. Hart, and R. Duda. A functional approach to integrating database and expert systems. Communications of the ACM, December 1988, 31 (12), 1424-1437.

Robinson, J. A. A machine-oriented logic based on the resolution principle. Journal of the ACM, 1965, 12, 23-41.

Sagie, I. Computer-aided modeling and planning (CAMP). ACM Transactions on Mathematical Software, September 1986, 12 (3), 225-248.

Sen, A. Decision support systems: an activity-oriented design. Journal of Information Science, 1983, 7, 23-30.

Shaw, M. J., P. L. Tu, and P. De. Applying machine learning to model management in decision support systems. Decision Support Systems, 1988, 4, 285-305.

Sprague, R. H. and E. D. Carlson. Building Effective Decision Support Systems. Englewood Cliffs, New Jersey: Prentice-Hall, 1982.

Sprague, R. H. A framework for the development of decision support systems. MIS Quarterly, December 1980, 1-26.

Teorey, T. J., D. Yang, and J. P. Fry. A logical design methodology for relational databases using the extended entity-relationship model. ACM computing Surveys, June 1986, 18 (2), 197-222.

Ullman, J. Implementation of logical query languages for databases. ACM Transactions on Database Systems, September 1985, 10 (3), 289-321.

Wang, M. S., and J. F. Courtney. A conceptual architecture for generalized decision support system software. IEEE Transactions on Systems, Man, and Cybernetics, 1984, SMC-14 (5), 701-711.

Williams, H. P. Model Building in Linear and Integer Programming. In K. Schittkowski (Ed.) Computational Mathematical Programming. Berlin, Heidelberg: Springer-Verlag, 1985a.

_____. Model Building in Mathematical Programming. New York: John Wiley & Sons, 1985b.

APPENDIX A

THE EXAMPLE OF THE TARIFF RATES DSM

The tariff rates DSM is adopted from Williams (1985b) to compare a functional DSM with the one in a MAGIC, a ML. A power station is committed to meeting the electricity load demands over a day (Table 37). In addition to the estimated load demands there must be sufficient generators working at any time to allow an increase in load of up to 15 percent. This increase would have to be accomplished by adjusting the output generators already operating within their permitted limits.

Table 37.--Electricity Load Demands Over a Day

Time Periods Over a Day	Number of Hours	Electricity Load Demands (MW)
12 p.m. to 6 a.m.	6	15000
6 a.m. to 9 a.m.	3	30000
9 a.m. to 3 p.m.	6	25000
3 p.m. to 6 p.m.	3	40000
6 p.m. to 12 p.m.	6	27000

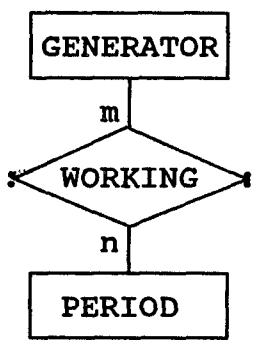
Three types of generators are available. Each generator has to work between a minimum and a maximum level. To start up a generator involves a cost. There is an hourly cost of running each generator at minimum level; there is also an additional hourly cost for each megawatt

(MW) above the minimum level. All these information is given in the Table 38 (with costs in dollars).

Table 38.-- Information of the Tariff Rates DSM

	Attribute Name			
Generator Type	Generator	1	2	3
Number of Units Available	Gu_Available	12	10	5
Minimum Level	Min_Level	850	1250	1500
Maximum Level	Max_Level	2000	1750	4000
Hourly Cost at Minimum Level	Costph	1000	2600	3000
Hourly Cost Per MW above Minimum	Excostph	2.0	1.3	3.0
Start-up Cost	Scostpu	2000	1000	500

The ERD, functional description and attributes of the tariff rates DSM are presented in Figure 47, Tables 39a and 39b, respectively. The LP formulation is presented in Figure 48. Due to the complexity of the DSM, the constraints are furthered explained in Table 40.



GENERATOR (Generator, Gu_Available, Min_Level, Max_Level, Costph, Excostph, Scostpu)

WORKING (Period, Generator, Number, Nst, Out)

PERIOD (Period, Hours, Load_Demand)

Figure 47. The ERD for the Tariff Rates DSM

Table 39a.--The Complete Functional Description of the Tariff Rates DSM

The Macro-level Description: a DSM Predicate

```
DSM (Tariff, [Upload, Hours, Load_Demand, Gu_Available,
      Min_Level, Max_Level, Costph, Excostph, Scostpu],
      [Number, Nst, Out], [min Total_Op_Cost], All)
```

The Micro-level Representation: a Functional Data Base

```
I. The IDB:
      UPLOAD (Upload)
      HOURS (Period, Hours)
      LOAD_DEMAND (Period, Load_Demand)
      GU_AVAILABLE (Generator, Gu_Available)
      MIN_LEVEL (Generator, Min_Level)
      MAX_LEVEL (Generator, Max_Level)
      COSTPH (Generator, Costph)
      EXCOSTPH (Generator, Excostph)
      SCOSTPU (Generator, Scostpu)
      NUMBER (Period, Generator, Number)
      NST (Period, Generator, Nst)
      OUT (Period, Generator, Out)
      Total_Op_Cost  = Total_Min_Cost + Total_Ex +
                       Total_Start_Cost
      Total_Min_Cost = + of (Min_Cost)
      Min_Cost       = Costph * Hours * Number
      Total_Ex       = + of (Ex_cost)
      Ex_Cost        = Excostph * Hours * (Out - Min_Out)
      Min_Out        = (Min_Level) * (Number)
      Total_Start_Cost    = + of (Start_Cost)
      Start_Cost     = Scostpu * Nst
      Period_Out     = + of (Out) by (Period)
      Period_Max_Out = + of (Max_Out) by (Period)
      Max_Out        = (Max_Level) * (Number)
      Extra_Demand   = (Upload) * (Load_Demand)
      Num_Increased  = (Number) - (Pre_Number)
      Pre_Number     = (pre) of (Number) order (Period) by
                       (Generator)
      All = (Period_Out ≥ Load_Demand) AND
          (Period_Max_Out ≥ Extra_Demand) AND
          (Out ≥ Min_Out) AND (Out ≤ Max_Out) AND
          (Nst ≥ Num_Increased) AND
          (Number ≤ Gu_Available) AND (Nst ≤ Gu_Available)
          AND (Number ≥ 0) AND (Nst ≥ 0) AND (Out ≥ 0)
```

Table 39a.--The Complete Functional Description of the Tariff Rates DSM (Continued)

The Micro-level Representation: A Functional Data Base

II. The EDB:

HOURS

(Period,	Hours)
1	6
2	3
3	6
4	3
5	6

LOAD_DEMAND

(Period,	Load_Demand)
1	15000
2	30000
3	25000
4	40000
5	27000

GU_AVAILABLE

(Generator,	Gu_Available)
G1	12
G2	10
G3	5

COSTPH

(Generator,	Costph)
G1	1000
G2	2600
G3	3000

MIN_LEVEL

(Generator,	Min_Level)
G1	850
G2	1250
G3	1500

MAX_LEVEL

(Generator,	Max_Level)
G1	2000
G2	1750
G3	4000

EXCOSTPH

(Generator,	Excostph)
G1	2
G2	1.3
G3	3

SCOSTPU

(Generator,	Scostpu)
G1	2000
G2	1000
G3	500

NUMBER

(Period,	Generator,	Number)	Corresponding Decision Variables
1	G1	-	X_{11}
2	G1	-	X_{21}
3	G1	-	X_{31}
4	G1	-	X_{41}
5	G1	-	X_{51}
1	G2	-	X_{12}
2	G2	-	X_{22}
3	G2	-	X_{32}
4	G2	-	X_{42}
5	G2	-	X_{52}
1	G3	-	X_{13}
2	G3	-	X_{23}
3	G3	-	X_{33}
4	G3	-	X_{43}
5	G3	-	X_{53}

Table 39a.--The Complete Functional Description of the Tariff Rates DSM (Continued)

II. The EDB (Continued):

NST (Period, Generator, Nst)			Corresponding Decision Variables
1	G1	-	Y_{11}
2	G1	-	Y_{21}
3	G1	-	Y_{31}
4	G1	-	Y_{41}
5	G1	-	Y_{51}
1	G2	-	Y_{12}
2	G2	-	Y_{22}
3	G2	-	Y_{32}
4	G2	-	Y_{42}
5	G2	-	Y_{52}
1	G3	-	Y_{13}
2	G3	-	Y_{23}
3	G3	-	Y_{33}
4	G3	-	Y_{43}
5	G3	-	Y_{53}

OUT (Period, Generator, Out)			Corresponding Decision Variables
1	G1	-	Z_{11}
2	G1	-	Z_{21}
3	G1	-	Z_{31}
4	G1	-	Z_{41}
5	G1	-	Z_{51}
1	G2	-	Z_{12}
2	G2	-	Z_{22}
3	G2	-	Z_{32}
4	G2	-	Z_{42}
5	G2	-	Z_{52}
1	G3	-	Z_{13}
2	G3	-	Z_{23}
3	G3	-	Z_{33}
4	G3	-	Z_{43}
5	G3	-	Z_{53}

Table 39b.--Attributes of The Tariff Rate DSM

Attributes	Type	Interpretations
*** Symbolic attributes:**		
Period	symbolic	time period over a day
Generator	symbolic	type of each generator
*** Parameter attributes:**		
Upload	numeric	percentage of reserve output guarantee in electricity load
Hours	numeric	number of hours in a period
Load_Demand	numeric	electricity load demand of a period
Gu_Available	numeric	number of units available of each generator type
Min_Level	numeric	minimum operation level of each generator type
Max_Level	numeric	maximum operation level of each generator type
Costph	numeric	hourly cost of each generator type at minimum operation level
Excostph	numeric	extra hourly cost of each generator type for every MW above the minimum level
Scostpu	numeric	start-up cost of each generator type
*** Variable attributes:**		
Number	numeric	number of units of each generator type working during a period
Nst	numeric	number of units of each generator type started up during a period
Out	numeric	total output rate from a unit of each generator type during a period.
*** Virtual attributes:**		
Total_Op_Cost	numeric	total operation cost
Total_Ex	numeric	total extra cost above the minimum level (lines 1-8 in Figure 48)
Ex_Cost	numeric	extra cost above the minimum level in a period
Min_Out	numeric	Minimum output with working units of each generator type

Table 39b.--Attributes of The Tariff Rate DSM (Continued)

Attributes	Type	Interpretations
Total_Min_Cost	numeric	total cost of all generators at minimum level (lines 9 -11 in Figure 48)
Min_Cost	numeric	cost of generators at minimum level in a period
Total_Start_Cost	numeric	total cost of starting up generators (lines 12-14 in Figure 48)
Start_Cost	numeric	cost of starting up each generator type in each period
Period_Out	numeric	electricity load generated in a period
Period_Max_Out	numeric	maximum of the electricity generated in a period
Max_Out	numeric	maximum output with generators of each type working
Extra_Demand	numeric	extra security load requirement guaranteed
Num_Increased	numeric	increase in number of generators of each type started up in a period
Pre_Number	numeric	number of generators of each type started up in the previous period
All	logical	constraints of the Tariff Rate DSM; they are explained in Table 40.

Minimize

$$
\begin{aligned}
&12\,(Z_{11} - 850\,X_{11}) + 6\,(Z_{21} - 850\,X_{21}) + && (1)\\
&12\,(Z_{31} - 850\,X_{31}) + 6\,(Z_{41} - 850\,X_{41}) + && (2)\\
&12\,(Z_{51} - 850\,X_{51}) + 7.8\,(Z_{12} - 1250\,X_{12}) + && (3)\\
&3.9\,(Z_{22} - 1250\,X_{22}) + 7.8\,(Z_{32} - 1250\,X_{32}) + && (4)\\
&3.9\,(Z_{42} - 1250\,X_{42}) + 7.8\,(Z_{52} - 1250\,X_{52}) + && (5)\\
&18\,(Z_{13} - 1500\,X_{13}) + 9\,(Z_{23} - 1500\,X_{23}) + && (6)\\
&18\,(Z_{33} - 1500\,X_{33}) + 9\,(Z_{43} - 1500\,X_{43}) + && (7)\\
&18\,(Z_{53} - 1500\,X_{53}) + && (8)\\
&6000\,X_{11} + 3000\,X_{21} + 6000\,X_{31} + 3000\,X_{41} + 6000\,X_{51} + && (9)\\
&15600\,X_{12} + 7800\,X_{22} + 15600\,X_{32} + 7800\,X_{42} + 15600\,X_{52} + && (10)\\
&18000\,X_{13} + 9000\,X_{23} + 18000\,X_{33} + 9000\,X_{43} + 18000\,X_{53} + && (11)\\
&2000\,Y_{11} + 2000\,Y_{21} + 2000\,Y_{31} + 2000\,Y_{41} + 2000\,Y_{51} + && (12)\\
&1000\,Y_{12} + 1000\,Y_{22} + 1000\,Y_{32} + 1000\,Y_{42} + 1000\,Y_{52} + && (13)\\
&500\,Y_{13} + 500\,Y_{23} + 500\,Y_{23} + 500\,Y_{43} + 500\,Y_{53} && (14)
\end{aligned}
$$

Subject to

$$
\begin{aligned}
&Z_{11} + Z_{12} + Z_{13} \geq 15{,}000 && (15)\\
&Z_{21} + Z_{22} + Z_{23} \geq 30{,}000 && (16)\\
&Z_{31} + Z_{32} + Z_{33} \geq 25{,}000 && (17)\\
&Z_{41} + Z_{42} + Z_{43} \geq 40{,}000 && (18)\\
&Z_{51} + Z_{52} + Z_{53} \geq 27{,}000 && (19)\\
&2000\,X_{11} + 1750\,X_{12} + 4000\,X_{13} \geq 17250 && (20)\\
&2000\,X_{21} + 1750\,X_{22} + 4000\,X_{23} \geq 34500 && (21)\\
&2000\,X_{31} + 1750\,X_{32} + 4000\,X_{33} \geq 28750 && (22)\\
&2000\,X_{41} + 1750\,X_{42} + 4000\,X_{43} \geq 46000 && (23)\\
&2000\,X_{51} + 1750\,X_{52} + 4000\,X_{53} \geq 31050 && (24)\\
&Z_{i1} - 850 * X_{i1} \geq 0 \quad \text{for } i=1 \text{ to } 5 && (25)\\
&Z_{i2} - 1250 * X_{i2} \geq 0 \quad \text{for } i=1 \text{ to } 5 && (26)\\
&Z_{i3} - 1500 * X_{i3} \geq 0 \quad \text{for } i=1 \text{ to } 5 && (27)\\
&Z_{i1} - 2000 * X_{i1} \leq 0 \quad \text{for } i=1 \text{ to } 5 && (28)\\
&Z_{i2} - 1750 * X_{i2} \leq 0 \quad \text{for } i=1 \text{ to } 5 && (29)\\
&Z_{i3} - 4000 * X_{i3} \leq 0 \quad \text{for } i=1 \text{ to } 5 && (30)\\
&Y_{11} - X_{11} + X_{51} \geq 0 && (31)\\
&Y_{i1} - X_{i1} + X_{(i-1)1} \geq 0 \quad \text{for } i=2 \text{ to } 5 && (32)\\
&Y_{12} - X_{12} + X_{52} \geq 0 && (33)\\
&Y_{i2} - X_{i2} + X_{(i-1)2} \geq 0 \quad \text{for } i=2 \text{ to } 5 && (34)\\
&Y_{13} - X_{13} + X_{53} \geq 0 && (35)\\
&Y_{i3} - X_{i3} + X_{(i-1)3} \geq 0 \quad \text{for } i=2 \text{ to } 5 && (36)\\
&X_{i1} \leq 12 \quad \text{for } i=1 \text{ to } 5 && (37)\\
&X_{i2} \leq 10 \quad \text{for } i=1 \text{ to } 5 && (38)\\
&X_{i3} \leq 5 \quad \text{for } i=1 \text{ to } 5 && (39)\\
&Y_{i1} \leq 12 \quad \text{for } i=1 \text{ to } 5 && (40)\\
&Y_{i2} \leq 10 \quad \text{for } i=1 \text{ to } 5 && (41)\\
&Y_{i3} \leq 5 \quad \text{for } i=1 \text{ to } 5 && (42)
\end{aligned}
$$

For all i=1 to 5, j= 1 to 3,
X_{ij}, Y_{ij}, $Z_{ij} \geq 0$ and X_{ij}, Y_{ij} are integers

Figure 48. The LP Formulation for the Tariff Rates DSM

Table 40.--Constraints of The Tariff Rate DSM

Conditions	Interpretations
(Period_Out ≥ Load_Demand)	electricity load demands must be met in each period (lines 15 -19 in Figure 48)
(Period_Max_Out ≥ Extra_Demand)	the extra guaranteed load requirement must be able to be met without starting up any more generators (lines 20-24).
(Out ≥ Min_Out) AND (Out ≤ Max_Out)	output must lie within the limits limits of the generators working. (lines 25-30 in Figure 48)
(Nst ≥ Num_Increased)	the number of generators started up in a period must equal the increase in number (lines 31 -36 in Figure 48).
(Number ≤ Gu_Available) AND (Nst ≤ Gu_Available)	the number of generators of a type working or stared up in each period must be bound to the total number of generators of each type (lines 37-42 in Figure 48).

APPENDIX B

THE COMPLETE PROLOG PROGRAM: TRANSLATOR

```
/**********   DOMAIN DECLARATION (omitted)  *********/

/************   DATABASE DECLARATION   **************/
DATABASE - workbase
   cons(boolean, string, string)
   current_num(symbol, integer)
   ofunc(opt, symbol)
   unknown(symbol)
DATABASE - modelbase
   dsm(symbol, inputs, outputs, objs, constraint)
DATABASE - idb
   relation(symbol, slist, slist)
   ldefn(string, conlist)
   defn(symbol, sclist)
DATABASE - edb
   tuple(symbol, slist,  slist)

/*******   PREDICATE DECLARATION (omitted)  *********/

/******************  THE GOAL     ******************/
GOAL
   go.

/***********  DEFINING ALL THE PREDICATES  **********/
CLAUSES
datapair(Rela, [Key, Data]) :- tuple(Rela, Key, Data).
```

```
/*----------   THE MAIN PREDICATE    ----------------*/
go :-
   windowsetup("Translating a Functional DSM"),
   read_mb(Mb),
   consult(Mb, modelbase),
   read_dsm(Dsm),
   read_idb(Fdb),
   read_edb(Data),
   read_output(DsmFile),
   concat("Translating the DSM \"", Dsm, Msg),
   concat(Msg, "\" (by Lijen Ko)", Msg2),
   windowsetup(Msg2),
   dsm(Dsm, _, Outputs, Objs, Constraint),
   field_str(1, 5, 45,
      "Step 1. Reading the functional data base."),
   consult(Fdb, idb),
   consult(Data, edb),
   !,
   field_str(3, 5, 45,
      "Step 2. Generating the decision variables."),
   dv_gen(Outputs),
   !,
   field_str(5, 5, 45,
      "Step 3. Generating objective function(s)."),
   obj_gen(Objs),
   !,
   field_str(7, 5, 45, "Step 4. Generating constraints."),
   constraint_gen(Constraint),
   !,
   field_str(9, 5, 45,
      "Step 5. Writing the output file."),
   convert(DsmFile),
   clear,
   tail_message(Dsm, DsmFile, I),
   I='y', go.
go :- exit.

/*-------------   LAYOUT A SCREEN  -----------------*/
/* Clear the screen and make a display.             */
windowsetup(Msg) :-
   makewindow(2, 62, 62, "Message", 19, 0, 6, 80),
   makewindow(1, 29, 29, Msg, 0, 0, 19, 80).

/*------    READ A FILE NAME OF THE MODEL BASE  ----*/
/* Ask for the file name of the model base.         */
read_mb(Name) :-
   repeat,
   field_str(1, 5, 33,
   "Please enter the model base name."),
   cursor(1, 40),
   readln(Name),
   str_clist(Name, L),
```

```
   not (ill_name(L)),
   filexist(Name).

/*------    READ THE NAME OF A FUNCTIONAL DSM    ----*/
/* Ask for the name of the functional dsm.          */
read_dsm(Name) :-
   repeat,
   cursor(3, 5),
   write("Please enter the name of the decision support
      model."),
   cursor(3, 59),
   readln(Name),
   str_clist(Name, L),
   not (ill_name(L)),
   dsmexist(Name).

/*--------   READ THE FILE NAME OF AN IDB   --------*/
/* Ask for the file name of an IDB.                 */
read_idb(Name) :-
   repeat,
   cursor(5, 5),
   write("Please enter the name of an IDB.  "),
   cursor(5, 39),
   readln(Name),
   str_clist(Name, L),
   not (ill_name(L)),
   filexist(Name).

/*--------   READ THE FILE NAME OF AN EDB   --------*/
/* Ask for the file name of an EDB.                 */
read_edb(Name) :-
   repeat,
   cursor(7, 5),
   write("Please enter the name of an EDB.  "),
cursor(7, 39),
   readln(Name),
   str_clist(Name, L),
   not (ill_name(L)),
   filexist(Name).

/*--------   READ THE NAME OF THE OUTPUT FILE ------*/
/* Ask for the name of the output file.             */
read_output(DsmFile) :-
   repeat,
   cursor(9, 5),
   write("Please enter the name of the output file. "),
   cursor(9, 48),
   readln(DsmFile),
   str_clist(DsmFile, L),
   not (ill_name(L)),
   output_ok(DsmFile).
```

```
/*----------  GENERATE DECISION VARIABLES  ---------*/
dv_gen([]).
dv_gen([Dv|Dvs]) :-
   assertz(unknown(Dv), workbase),
   dv_gen1(Dv),
   dv_gen(Dvs).

dv_gen1(Dv) :-
   retract (tuple(Dv, X, []), edb),
   gensym(Dv, Dv_new),
   assertz (tuple(Dv, X, [Dv_new]), edb),
   fail.
dv_gen1(_).

/*----------  GENERATE OBJECTIVE FUNCTIONS ---------*/
obj_gen([]).
obj_gen([obj_func(Opt, Obj)|Objs]) :-
   interpret(Obj),
   assertz(ofunc(Opt, Obj)),
   obj_gen(Objs).

/*----------  GENERATE CONSTRAINTS      ------------*/
constraint_gen("").
constraint_gen(Constraint) :-
   ldefn(Constraint, Conlist),
   constraint_gen1(Conlist).

constraint_gen1([]) :- !.
constraint_gen1([c(Attr1, Boolean, "zero")|Conds]) :-
   interpret(Attr1),
   constraint_gen2 (Boolean, Attr1),
   constraint_gen1 (Conds).
constraint_gen1([c(Attr1, Boolean, Attr2)|Conds]) :-
   interpret(Attr1),
   interpret(Attr2),
   relation(Attr1, Key1, _),  /* check comparability  */
   relation (Attr2, Key2, _),
   set_equal(Key1, Key2),
   constraint_gen3 (Boolean, Attr1, Attr2),
   constraint_gen1 (Conds).

constraint_gen2 (Boolean, Attr1) :-
   tuple (Attr1, _, [Data1]),
   assertz(cons(Boolean, Data1, "0")),
   fail.
constraint_gen2 (_, _).

constraint_gen3 (Boolean, Attr1, Attr2) :-
   tuple (Attr1, Key1, [Data1]),
   tuple (Attr2, Key2, [Data2]),
   set_equal(Key1, Key2),
   assertz(cons(Boolean, Data1, Data2)),
```

```
   fail.
constraint_gen3 (_, _, _).

/*----------  GENERATE THE MATHEMATICAL DSM  -------*/
convert(DsmFile) :-
   openwrite(dsmfile, DsmFile),
   writedevice(dsmfile),
   wrt_obj,
   wrt_consnt,
   writedevice(screen),
   closefile(dsmfile),
   !.

/*-  WRITE OBJECTIVE FUNCTIONS TO THE OUTPUT FILE --*/
wrt_obj :-
   retract(ofunc(Opt, Obj)),
   tuple(Obj, [], [ObjF]),
   write(Opt, "\t", ObjF, "\n"),
   fail.
wrt_obj :- nl.

/*-  WRITE OBJECTIVE FUNCTIONS TO THE OUTPUT FILE --*/
wrt_consnt :-
   retract(cons(Boolean, Data1, Data2)),
   write("subject to \n"),
   logic_op(Boolean, Lop),
   write("\t", Data1, "\t", Lop, "\t", Data2, "\n"),
   wrt_consnt1.
wrt_consnt.

wrt_consnt1 :-
   retract(cons(Boolean, Data1, Data2)),
   logic_op(Boolean, Lop),
   write("\t", Data1, "\t", Lop, "\t", Data2, "\n"),
   fail.
wrt_consnt1.

/*---------  CLEAR THE WORKING DATA BASES.  -------*/
clear :-
  retractall(cons(_, _, _)),
  retractall(current_num(_, _)),
  retractall(ofunc(_, _)),
  retractall(unknown(_)),
  retractall(dsm(_, _, _, _, _)),
  retractall(defn(_, _)),
  retractall(ldefn(_, _)),
  retractall(relation(_, _, _)),
  retractall(tuple(_, _, _)),
  !.

/* DISPLAY A MESSAGE AT THE END OF THE TRANSLATION. */
tail_message(Dsm, DsmFile, I) :-
```

```
   shiftwindow(2),
   clearwindow,
   cursor(0, 5),
   write("The translation of the DSM ", Dsm, " is
      finished."),
   cursor(1, 5),
   write("The output is stored in the file \"",
      DsmFile, "\"."),
   cursor(3,5),
   write("Do you want to translate another DSM? (y/n)  "),
   cursor(3, 50),
   readchar(L),
   upper_lower(L, I).

/******** INTERPRET A NUMERIC ATTRIBUTE ***********/
interpret(Attr) :- interpreted(Attr), !.
interpret(Attr) :-
   defn(Attr, Exp),
   interpret(_, Attr, Exp).

/*********** EXPRESSION INTERPRETATION ***********/
/*--------------- scalar operation ----------------*/
/* <expression> ::= <na> + <na> |  <na> - <na>  |   */
/*                  <na> * <na> |  <na> / <na>      */
interpret(1, Va, [r(Attr1), o(Op), r(Attr2)]) :-
   interpret(Attr1),
   interpret(Attr2),
   product(Attr1, Attr2, New_Rela),
   cmp(Attr1, Op, Attr2, New_Rela, Va),
   retract(relation(New_Rela, Key, _), idb),
   assertz(relation(Va, Key, [Va]), idb).

/*---------------- simple reduction ---------------*/
/*  <op> of (<na>)                                 */
interpret(2, Va, [o(Op), s(of), r(Attr)]) :-
   interpret(Attr),
   findall (Data, tuple(Attr, _, [Data]), Data_List),
   sreduce(Va, Op, Attr, Data_List, Result),
   assertz(relation(Va, [], [Va]), idb),
   assertz(tuple(Va, [], [Result]), edb).

/*--------------- equivalence reduction ----------*/
/*    <op> of (<na>) by (<ca>)                    */
interpret(3, Va,
[o(Op), s(of), r(Attr), s(by), l(Ctrl)]) :-
   interpret(Attr),
   relation (Attr, KeyA, _),
   seq (KeyA, Ctrl, C_Seq),
   findall (Datapair, datapair(Attr, Datapair), List),
   modify3(List, CList, C_Seq),
   my_sort(CList, [H|Rest], []),
   ereduce (Attr, Va, Op, H, Rest),
```

```
   assertz(relation(Va, Ctrl, [Va]), idb).

/*---------------- functional mapping --------------*/
/*   <op> of (<na>) order (<oa>)                    */
interpret (4, Va,
[o(Op), s(of), r(Attr), s(order), l(Ord)]) :-
   interpret(Attr),
   relation (Attr, KeyA, _),
   seq (KeyA, Ord, Seq),
   findall (Datapair, datapair(Attr, Datapair), List),
   modify3(List, OList, Seq),
   my_sort(OList, [H|Rest], []),
   freduce (Attr, Va, Op, H, Rest),
   assertz(relation(Va, Ord, [Va]), idb).

/*---------- partial functional mapping ------------*/
/*  <op> of (<na>) order (<oa>) by (<ca>)           */
interpret (5, Va,
[o(Op),s(of),r(Attr),s(order),l(Ord),s(by),l(Ctrl)]) :-
   interpret(Attr),
   relation (Attr, KeyA, _),
   seq (KeyA, Ctrl, C_Seq),
   seq (KeyA, Ord, O_Seq),
   findall (Datapair, datapair(Attr, Datapair), List),
   modify5(List, NewList, C_Seq, O_Seq),
   my_sort2 (NewList, [H|Rest], []),
   pfreduce (Attr, Va, Op, H, Rest),
   append (Ctrl, Ord, KeyA2),
   assertz(relation(Va, KeyA2, [Va]), idb).

/******    SUBROUTINES USED IN "INTERPRET"   ********/
/*--------------   scalar operation   --------------*/
cmp(Attr1, Op, Attr2, New_Rela, Va) :-
   known(Attr1),
   known(Attr2), !,
   compute(New_Rela, Op, Va).
cmp(_, Op, _, New_Rela, Va) :-
   assertz(unknown(Va), workbase),
   transform(New_Rela, Op, Va).

compute (New_Rela, Op, Va) :-
   retract(tuple(New_Rela, Key_list, [Data1, Data2]),
      data),
   str_real(Data1, D1),
   str_real(Data2, D2),
   evaluate11 (Op, D1, D2, R),
   str_real(Result, R),
   assertz (tuple(Va, Key_list, [Result]), edb),
   fail.
compute (_, _, _).

transform (New_Rela, Op, Va) :-
```

```
    retract(tuple (New_Rela, Key_list, [Data1, Data2]),
    edb),
    concat("(", Data1, Dd1),
    concat(Dd1, ")", D1),
    concat("(", Data2, Dd2),
    concat(Dd2, ")", D2),
    concat(D1, Op, Temp),
    concat(Temp, D2, Result),
    assertz (tuple (Va, Key_list, [Result]), edb),
    fail.
transform (_, _, _).

/*------------  simple reduction  -----------------*/
sreduce(_, Op, Attr, Data_List, Result) :-
    known(Attr), !,
    evaluate(Op, Data_List, R),
    str_real(Result, R).
sreduce(Va, Op, _, Data_List, Result) :-
    assertz(unknown(Va), workbase),
    evaluate2(Op, Data_List, Result).

/*-----------  equivalence reduction  -------------*/
ereduce(Attr, Va, Op, H, Rest) :-
    known(Attr), !,
    equ_reduce (Va, Op, H, Rest).
ereduce(_, Va, Op, H, Rest) :-
    assertz(unknown(Va), workbase),
    equ_reduce2 (Va, Op, H, Rest).

equ_reduce (Va, _, [_, C2, Sofar], []) :-
    assertz(tuple(Va, C2, Sofar), edb).
equ_reduce (Va, Op, [A1, C1, [Sofar]], [[A1, C1,
[Data]]|Others]) :-
    !,
    eval11(Op, Sofar, Data, Temp),
    equ_reduce (Va, Op, [A1, C1, [Temp]], Others).
equ_reduce (Va, Op, [_, C1, Sofar], [[A2, C2,
D2]|Others]) :-
    !,
    assertz(tuple(Va, C1, Sofar), edb),
    equ_reduce (Va, Op, [A2, C2, D2], Others).

equ_reduce2 (Va, _, [_, C2, Sofar], []) :-
    assertz(tuple(Va, C2, Sofar), edb).
equ_reduce2 (Va, Op, [A1, C1, [Sofar]], [[A1, C1,
[Data]]|Others]) :-
    !,
    concat(Sofar, Op, Tmp),
    concat(Tmp, Data, Temp),
    equ_reduce2 (Va, Op, [A1, C1, [Temp]], Others).
equ_reduce2 (Va, Op, [_, C1, Sofar], [[A2, C2,
D2]|Others]) :-
```

```
   !,
   assertz(tuple(Va, C1, Sofar), edb),
   equ_reduce2 (Va, Op, [A2, C2, D2], Others).

/*-----------   functional mapping     -------------*/
freduce (Attr, Va, Op, H, Rest) :-
   known(Attr), !,
   func_reduce (Va, Op, H, Rest).
freduce (_, Va, Op, H, Rest) :-
   assertz(unknown(Va), workbase),
   func_reduce2 (Va, Op, H, Rest).

func_reduce (Va, _, [_, O2, Sofar], []) :-
   assertz(tuple(Va, O2, Sofar), edb).
func_reduce (Va, Op, [A1, O1, [Sofar]], [[A1, O1,
[Data]]|Os]) :-
   !,
   eval11(Op, Sofar, Data, Temp),
   func_reduce (Va, Op, [A1, O1, [Temp]], Os).
func_reduce (Va, Op, [_, O1, [Sofar]], [[A2, O2,
[D2]]|Os]) :-
   !,
   assertz(tuple(Va, O1, [Sofar]), edb),
   eval11(Op, Sofar, D2, Temp),
   func_reduce (Va, Op, [A2, O2, [Temp]], Os).

func_reduce2 (Va, _, [_, O2, Sofar], []) :-
   assertz(tuple(Va, O2, Sofar), edb).
func_reduce2 (Va, Op, [A1, O1, [Sofar]], [[A1, O1,
[Data]]|Os]) :-
   !,
   concat(Sofar, Op, Tmp),
   concat(Tmp, Data, Temp),
   func_reduce2 (Va, Op, [A1, O1, [Temp]], Os).
func_reduce2 (Va, Op, [_, O1, [Sofar]], [[A2, O2,
[D2]]|Os]) :-
   assertz(tuple(Va, O1, [Sofar]), edb),
   concat(Sofar, Op, Tmp),
   concat(Tmp, D2, Temp),
   func_reduce2 (Va, Op, [A2, O2, [Temp]], Os).

/*------------  partial functional mapping ---------*/
pfreduce (Attr, Va, Op, H, Rest) :-
   known(Attr), !,
   pfunc(Va, Op, H, Rest).
pfreduce (_, Va, Op, H, Rest) :-
   assertz(unknown(Va), workbase),
   pfunc2 (Va, Op, H, Rest).

pfunc (Va, _, [_, Key, Sofar], []) :-
   !,
   assertz(tuple(Va, Key, Sofar), edb).
```

```
pfunc (Va, Op, [A, Key, [Sofar]], [[A, Key, [Data]]|Os])
:- !,
   eval11(Op, Sofar, Data, Temp),
   pfunc (Va, Op, [A, Key, [Temp]], Os).
pfunc (Va, Op, [[_, C1, _], K1, [Sofar]], [[[CO2, C1, O2],
K2, [Data]]|Os]) :-
   !,
   assertz(tuple(Va, K1, [Sofar]), edb),
   eval11(Op, Sofar, Data, Temp),
   pfunc (Va, Op, [[CO2, C1, O2], K2, [Temp]], Os).
pfunc (Va, Op, [_, K1, Sofar], [H|Os]) :-
   !,
   assertz(tuple(Va, K1, Sofar), edb),
   pfunc(Va, Op, H, Os).

pfunc2 (Va, _, [_, Key, Sofar], []) :-
   !,
   assertz(tuple(Va, Key, Sofar), edb).
pfunc2 (Va, Op, [A, Key, [Sofar]], [[A, Key, [Data]]|Os])
:- !,
   concat(Sofar, Op, Tmp),
   concat(Tmp, Data, Temp),
   pfunc2 (Va, Op, [A, Key, [Temp]], Os).
pfunc2 (Va, Op, [[_, C1, _], K1, [Sofar]], [[[CO2, C1,
O2], K2, [Data]]|Os]) :-
   !,
   assertz(tuple(Va, K1, [Sofar]), edb),
   concat(Sofar, Op, Tmp),
   concat(Tmp, Data, Temp),
   pfunc2 (Va, Op, [[CO2, C1, O2], K2, [Temp]], Os).
pfunc2 (Va, Op, [_, K1, Sofar], [H|Os]) :-
   !,
   assertz(tuple(Va, K1, Sofar), edb),
   pfunc2(Va, Op, H, Os).

/*************** SUBROUTINES  ***************/
dsmexist(Name) :- dsm(Name,  _, _, _, _), !.
dsmexist(Name) :-
   concat("There is no DSM with the name \"", Name, Msg),
   concat(Msg, "\".", Msg2),
   message(Msg2),
   fail.

element (_, 0, "")   :- !.
element ([], _, "")  :- !.
element ([A|_], 1, A) :- !.
element ([_|B], Num, C) :- N=Num-1, element(B, N, C).

eval11(Op, Data1, Data2, Result) :-
   str_real(Data1, D1),
   str_real(Data2, D2),
   evaluate11(Op, D1, D2, R),
```

```
   str_real(Result, R).

evaluate(_, [X], D) :-
   !,
   str_real(X, D).
evaluate(Op, [Data|X], R) :-
   str_real(Data, D),
   evaluate(Op, X, Sofar),
   evaluate11(Op, D, Sofar, R).

evaluate11("+", D1, D2, R) :- !, R = D1+D2.
evaluate11("-", D1, D2, R) :- !, R = D1-D2.
evaluate11("*", D1, D2, R) :- !, R = D1*D2.
evaluate11("/", D1, D2, R) :- !, R = D1/D2.

evaluate2(_, [X], X) :- !.
evaluate2(Op, [Data|X], Result) :-
   concat(Data, Op, Temp),
   evaluate2(Op, X, Sofar),
   concat(Temp, Sofar, Result).

filexist(Name) :- existfile(Name), !.
filexist(Name) :-
   concat("There is no file with the name \"",
      Name, Msg),
   concat(Msg, "\".", Msg2),
   message(Msg2),
      fail.

form (_, [], _, [], []) :- !.
form (List1, [N1|X1], List2, [0|X2], [A1|Z]) :-
   !,
   element (List1, N1, A1),
   form (List1, X1, List2, X2, Z).
form (List1, [0|X1], List2, [N2|X2], [A2|Z]) :-
   !,
   element (List2, N2, A2),
   form (List1, X1, List2, X2, Z).
form (List1, [N1|X1], List2, [N2|X2], [A1|Z]) :-
   !,
   element (List1, N1, A1),
   element (List2, N2, A2),
   A1=A2,
   form (List1, X1, List2, X2, Z).

ill_name([H|_]) :-
     period(H), !,
     message("Please begin the file name with a letter
     or digit.").
ill_name(L) :- ill_name2(L).

ill_name2([H|_]) :-
```

```
      not (legal_letter(H)), !,
      message("The file name contains illegal
         characters.").
ill_name2([_|B]) :- ill_name2(B).

interpreted(Attr) :- relation(Attr, _, _).

known(A) :- not (unknown(A)).

legal_letter(G) :- period(G).            /* '.' */
legal_letter(G) :- numeric(G), !.        /* 0-9 */
legal_letter(G) :- G>64, G<91, !.        /* A-Z */
legal_letter(G) :- G=95, !.              /* underscore */
legal_letter(G) :- G>96, G<123.          /* a-z */

logic_op(lt, "<").
logic_op(le, "<=").
logic_op(eq, "=").
logic_op(ne, "\=").
logic_op(gt, ">").
logic_op(ge, ">=").

/* Combine a list of strings into a string. (o, i)  */
make_atom ("", []) :- !.
make_atom (String, [K|Ks]) :-
make_atom (Sofar, Ks),
concat(K, Sofar, String).

message(Msg) :-
  gotowindow(2),
  clearwindow,
  field_str(1, 5, 60, Msg),
  field_str(2, 5, 26, "Press any key to continue."),
  readchar(_),
  gotowindow(1).

modify3([], [], _) :- !.
modify3([[Key, Data]|Rest], [[[Ctrl], C_Key, Data]|CRest],
Seq) :- swap (Key, C_Key, Seq),
   make_atom(Ctrl, C_Key),
   modify3(Rest, CRest, Seq).

modify5([], [], _, _) :- !.
modify5([[Key, D]|Rest], [[[A, A1, A2], Key2, D]|NewRest],
  CSeq, OSeq) :-
   swap (Key, CKey, CSeq),
      make_atom(A1, CKey),
      swap (Key, OKey, OSeq),
      make_atom(A2, OKey),
      concat (A1, A2, A),
      append (CKey, Okey, Key2),
      modify5(Rest, NewRest, CSeq, OSeq).
```

```
numeric(C) :- C>47, C<58.

/*---------------- order (A, L, O)  -----------------*/
/*  Find the order, O, of an atom A in the list L.  */
order(A, [A|_], 1).
order(A, [_|B], D) :- order(A, B, DD), D=DD+1.

/*--------------- order2 (A, L, O)  -----------------*/
/*  Find the position O, of an atom A in the list L.*/
/*  If not found, return 0 as the order.            */
order2(A, List, Order) :- order(A, List, Order).
order2(_, _, 0).

output_ok(Name) :- not(existfile(Name)), !.
output_ok(Name) :-
   shiftwindow(2),
   clearwindow,
   cursor(1, 5),
   concat("There is a file with the name \"", Name,
      Msg),
   concat(Msg, "\".  Overwrite it? (y/n)  ", Msg2),
   write(Msg2),
   readchar(L),
   shiftwindow(1),
   upper_lower(L, I),
   I = 'y'.

period(C) :- C=46.

/*-------------- product(R1, R2, R3) ----------------*/
/*  Compute the third relation as the Cartesian     */
/*     product of the first two.                    */
product(Rela1, Rela2, New_Rela) :-
   relation(Rela1, Key1, A1),
   relation(Rela2, Key2, A2),
   gensym(work, New_Rela),
   union(Key1, Key2, Key),
   append(A1, A2, Attrs),
   assertz(relation(New_Rela, Key, Attrs), idb),
   seq(Key1, Key, Sequ1),
   seq(Key2, Key, Sequ2),
   product1(Rela1, Sequ1, Rela2, Sequ2, New_Rela), !.

product1(Rela1, Sequ1, Rela2, Sequ2, New_Rela) :-
   tuple(Rela1, List1, Data1),
   product2(List1, Data1, Sequ1, Rela2, Sequ2, New_Rela),
   fail.
product1(_, _, _, _, _).

product2(List1, Data1, Sequ1, Rela2, Sequ2, New_Rela) :-
   tuple(Rela2, List2, Data2),
```

```
   form(List1, Sequ1, List2, Sequ2, List),
   append(Data1, Data2, Data),
   assertz(tuple(New_Rela, List, Data), edb),
   fail.
product2(_, _, _, _, _, _).

/*------------ seq  (L1, L2, Orders)  --------------*/
/* For each element in L2, find its position in L1  */
/*    and store the position in the list "Order".   */
seq(_, [], []).
seq(L1, [A|B], [Order|D]) :-
   order2(A, L1, Order),
   seq(L1, B, D).

/*------------- swap (Old, New, Seq) ---------------*/
/*    Swap elements in List1 as indicated in Seq.   */
swap (_, [], []) :- !.
swap (List, Clist, [0|Slist]) :-
   !, swap (List, Clist, Slist).
swap (List, [A|Clist], [N|Slist]) :-
   element (List, N, A),
   swap (List, Clist, Slist).

my_sort ([], X, X)    :- !.
my_sort ([H|T], S, X) :-
   split(H,T,A,B),
   my_sort (A, S, [H|Y]),
   my_sort (B, Y, X).

split([[A1]|B1], [[[A2]|B2]|X], [[[A2]|B2]|Y], Z) :-
   A2<A1, !, split([[A1]|B1], X, Y, Z).
split(H, [A|X], Y, [A|Z]) :-
   split(H, X, Y, Z).
split(_, [], [], []).

my_sort2 ([], X, X)    :- !.
my_sort2 ([H|T], S, X) :-
   split2(H,T,A,B),
   my_sort2 (A, S, [H|Y]),
   my_sort2 (B, Y, X).

split2([[A1 | CO1]|B1], [[[A2|CO2]|B2]|X],
[[[A2|CO2]|B2]|Y], Z) :-
   A2<A1, !, split2([[A1|CO1]|B1], X, Y, Z).
split2(H, [A|X], Y, [A|Z]) :-
   split2(H, X, Y, Z).
split2(_, [], [], []).

/*****************  PROLOG LIBRARY ******************/
append([],     L,  L).
append([A|L1], L2, [A|L3]) :- append(L1, L2, L3).
```

```
efface(A, [A|L], L) :- !.
efface(A, [B|L], [B|M]) :- efface(A, L, M).

gensym(Root, Atom):-
   get_num(Root, Num),
   str_int(Numname, Num),
   concat(Root, Numname, Atom).

get_num(Root, Num):-
   retract(current_num(Root, Num1), workbase), !,
   Num=Num1+1,
   asserta(current_num(Root, Num), workbase).
get_num(Root, 1):-
   asserta(current_num(Root, 1), workbase).

intersect([], _, []).
intersect([X|R], Y, [X|Z]) :-
     member(X, Y), !,
     intersect(R, Y, Z).
intersect([_|R], Y, Z) :-
     intersect(R, Y, Z).

member(X, [X|_]).
member(X, [_|Y]):- member(X, Y).

repeat.
repeat :- repeat.

set_equal([], []).
set_equal([A|B], [A|D]) :- !,
   set_equal(B, D).
set_equal([A|B], [C|D]) :-
   efface(A, D, E),
   set_equal(B, [C|E]).

str_clist("", []).
str_clist(S, [H|T]) :-
   frontchar(S, H, S1),
   str_clist(S1, T).

union([], X, X).
union([X|R], Y, Z) :-
   member(X, Y), !,
   union(R, Y, Z).
union([X|R], Y, [X|Z]) :-
   union(R, Y, Z).

/***************        THE END        ******************/
```

VITA

Lijen Ko was born in Taipei, Taiwan on July 22, 1957. In 1975, after graduating from Taipei First Girls' Senior High School, she was admitted to National Chung-Hsing University where she received a Bachelor of Science in Business Administration. From August 1979 to June 1980, she worked for Song-Kang Book Company as a programmer analyst. Then, she was employed by Taiwan Subsidiary of Grumman Data Systems Corporation as a system analyst. Later she was promoted to be a project supervisor. In 1983, she came to the United States to join her husband and started her graduate study in System Science at LSU that fall. She received a Master of Science in System Science in May 1986. After passing the comprehensive examination for her master degree in 1985, she began the pursuit of a Doctor of Philosophy at LSU with a major in Management Information Systems and a minor in Computer Science.

DOCTORAL EXAMINATION AND DISSERTATION REPORT

Candidate: Lijen Ko

Major Field: Business Administration

Title of Dissertation: A Functional Approach to Model Management

Approved:

Major Professor and Chairman

Dean of the Graduate School

EXAMINING COMMITTEE:

Date of Examination:

April 20, 1990